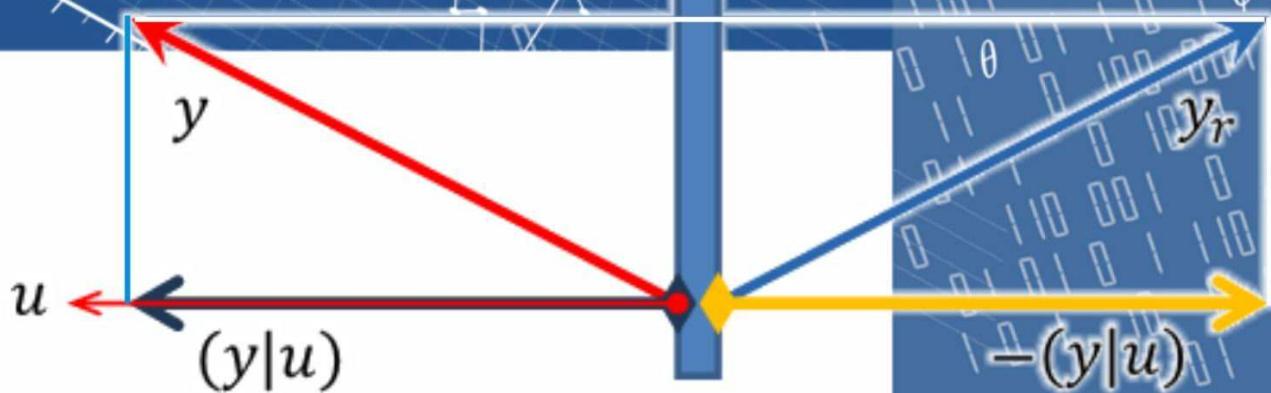


И.В. Семушин  
Ю.В. Цыганова  
В.В. Воронина

В.В. Угаров  
А.И. Афанасова  
И.Н. Куличенко



# Вычислительная линейная алгебра в проектах на C#

Ульяновск 2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

# ВЫЧИСЛИТЕЛЬНАЯ ЛИНЕЙНАЯ АЛГЕБРА В ПРОЕКТАХ НА С#

Учебное пособие

Ульяновск  
УлГТУ  
2014

УДК 519.61 + 519.654 + 519.244.4 (075.8)

ББК 22.193я7

В94

Рецензенты: д-р физ.-мат. наук, профессор Н. О. Седова, УлГУ,  
д-р техн. наук, профессор Н. Г. Ярушкина, УлГТУ,  
канд. физ.-мат. наук, доцент А. В. Цыганов, УлГПУ

Утверждено редакционно-издательским советом университета  
в качестве учебного пособия

**Вычислительная линейная алгебра в проектах на C#**: учебное  
пособие / И. В. Семушин, Ю. В. Цыганова, В. В. Воронина, В. В. Угаров,  
А. И. Афанасова, И. Н. Куличенко. — Ульяновск : УлГТУ, 2014. — 429 с.  
ISBN 978-5-9795-1342-3

Учебное пособие охватывает базовые алгоритмы вычислительной линейной алгебры (ВЛА) и ориентирует на их анализ и полномасштабное исследование методом проектов.

Предлагаемые авторами проекты содержат более 250 индивидуальных заданий по основным темам ВЛА в трёх частях: «Стандартный курс», «Повышенный курс» и «Специальный курс».

Учебное пособие предназначено для студентов и аспирантов, обучающихся на факультетах информационных и вычислительных технологий.

Подготовлено на кафедре «Информационные системы» университета.

**УДК 519.61 + 519.654 + 519.244.4 (075.8)**  
**ББК 22.193я7**

ISBN 978-5-9795-1342-3

© Колл. авторов, 2014  
© Оформление. УлГТУ, 2014

RUSSIAN FEDERATION'S MINISTRY FOR EDUCATION AND SCIENCE  
Federal State Educational Government-Financed Institution of  
Higher Professional Education  
**«ULYANOVSK STATE TECHNICAL UNIVERSITY»**

# COMPUTATIONAL LINEAR ALGEBRA BY C# PROJECTS

Study guide

Ulyanovsk  
UISTU  
2014

UDC 519.61 + 519.654 + 519.244.4 (075.8)

BBK 22.193я7

B94

Reviewers: N. O. Sedova, D.Sc. (Phys&Math), Professor, UISU, Ulyanovsk,

N. G. Yarushkina, D.Sc. (Eng), Professor UISTU, Ulyanovsk,

A. V. Tsyganov, C.Sc. (Phys&Math), Docent, UISPU, Ulyanovsk

Approved by the University Editorial Board

as a study guide

**Computational linear algebra by C# projects: Study guide /**

B94 I. V. Semushin, Yu. V. Tsyganova, V. V. Voronina, V. V. Ugarov, A. I. Afanasova,

I. N. Kulichenko. – Ulyanovsk : UISTU, 2014. – 429 p.

ISBN 978-5-9795-1342-3

The book covers the basic algorithms of Computational Linear Algebra (CLA) and aims students at algorithm analysis and full-scale study by the project method.

The projects offered by the authors contain more than 250 individual assignments on the basic CLA topics arranged into three parts: «Standard course», «Advanced course», and «Special course».

The book is intended for graduate or post graduate students majoring in information and computational technologies.

The book has been developed and finalized for publication at the Chair of Information Systems of the university.

**UDC 519.61 + 519.654 + 519.244.4 (075.8)**

**BBK 22.193я7**

ISBN 978-5-9795-1342-3

© Authors, 2014

© Design. UISTU, 2014

Тогда попросил учитель: Поведай нам об Учении.

И он сказал так:

Ни один человек не откроет вам больше, чем то, что дремлет  
уже в предрассветных проблесках вашего знания.

И тот учитель, что ходит в тени храма в окружении учеников,  
дарит им не мудрость свою, но лишь веру свою и влюблённость.

Если он мудр, то не запретит войти тебе в дом своей мудрости, а  
лишь подведёт к порогу твоего собственного ума.

Астроном может рассуждать с тобой о своём понимании  
космоса, но он не может дать тебе своё понимание.

Музыкант может петь тебе в ритмах, которыми полнится мир, но  
он не может дать тебе ни слуха, что ловит те ритмы, ни голоса,  
чтобы их повторить.

И тот, кто воспет в науке о числах, может поведать тебе о  
значениях мер и весов, но он не сможет провести тебя дальше.

Ибо прозрение человеческое не сдаёт взаймы свои крылья  
другому.

И как каждый из вас одинок в провидении Божьем, так и  
каждому быть одиноким в познании Бога и в понимании Земли.

—Халиль Джебран, Пророк

Великая книга Природы написана  
на математическом языке.

—Галилей

Then said a teacher, Speak to us of Teaching.

And he said:

No man can reveal to you aught but what already lies half asleep in  
the dawning of your knowledge.

The teacher who walks in the shadow of the temple, among his  
followers, gives not of his wisdom but rather of his faith and his  
lovingness.

If he is indeed wise he does not bid you enter the house of his  
wisdom, but rather leads you to the threshold of your own mind.

The astronomer may speak to you of his understanding of the space,  
but he cannot give you his understanding.

The musician may sing to you of the rhythm which is in all space, but  
he cannot give you the ear which arrests the rhythm nor the voice that  
echoes it.

And he who is versed in the science of numbers can tell you of the  
regions of weight and measure, but he cannot conduct you thither.

For the vision of the man lends not its wings to another man.

And even as each one of you stands alone in God's knowledge, so  
must each one of you be alone in his knowledge of God and in his  
understanding of the earth.

—Kahlil Gibran, The Prophet

**The great book of Nature was written  
in mathematics language.**

—Galilei

# **Признательность**

Авторы выражают признательность Российскому Фонду Фундаментальных исследований за частичное финансирование этой работы в форме научно-исследовательского гранта № 14-07-00665.

# Acknowledgment

Acknowledgment is given by the authors to the Russian Federation for Basic Research for partly funding this work by research grant № 14-07-00665.

# Оглавление

<b>Предисловие</b>	14
<b>Введение</b>	17
Учебные цели студента	17
Оценка работы студента	18
Кодекс студента	23
Краткое описание курса	25
<b>I ОСНОВАНИЯ</b>	27
<b>1 Проекто-ориентированная методика</b>	29
1.1 Метод проектов	29
1.2 Фронтально-состязательный подход (ФСП)	32
1.3 Оценивание качества академических проектов	39
1.4 Положения о выполнении и защите проектов	47
1.5 Заключение по разделу 1	51
<b>2 Элементы программирования на C#</b>	52
2.1 Синтаксис	52
2.2 Приложения	55
2.3 Сервис	70
2.4 Заключение по разделу 2	83
<b>II СТАНДАРТНЫЙ КУРС</b>	85
<b>3 Проект № 1 «Стандартные алгоритмы LU-разложения»</b>	87
3.1 Алгоритмы метода Гаусса	87
3.2 Выбор ведущего элемента	90
3.3 Компактные схемы	94
3.4 Алгоритмы метода Жордана	97
3.5 Вычисление обратной матрицы	101
3.6 Плохо обусловленные матрицы	107
3.7 Задание на лабораторный проект № 1	109
3.8 Варианты задания на лабораторный проект № 1	112
3.9 Методические рекомендации для проекта № 1	114

3.10	Тестовые задачи для проекта № 1 . . . . .	129
3.11	Заключение по разделу 3 . . . . .	134
<b>4</b>	<b>Проект № 2 «Разложения Холесского» . . . . .</b>	<b>135</b>
4.1	Положительно определённые матрицы . . . . .	135
4.2	Квадратные корни из $P$ и алгоритмы Холесского . . . . .	136
4.3	Программная реализация алгоритмов Холесского . . . . .	139
4.4	Разложение Холесского: $ijk$ -формы . . . . .	141
4.5	Разложение Холесского: алгоритмы окаймления . . . . .	143
4.6	Особенности хранения ПО-матрицы $P$ . . . . .	146
4.7	Задание на лабораторный проект № 2 . . . . .	147
4.8	Варианты задания на лабораторный проект № 2 . . . . .	151
4.9	Методические рекомендации для проекта № 2 . . . . .	151
4.10	Тестовые задачи для проекта № 2 . . . . .	163
4.11	Заключение по разделу 4 . . . . .	165
<b>5</b>	<b>Проект № 3 «Ортогональные преобразования» . . . . .</b>	<b>166</b>
5.1	Ортогональные матрицы и приложения . . . . .	166
5.2	Линейная задача наименьших квадратов . . . . .	168
5.3	Ортогональные матрицы и наименьшие квадраты . . . . .	169
5.4	Преобразование Хаусхолдера . . . . .	170
5.5	Шаг триангуляризации матрицы . . . . .	175
5.6	Решение треугольной системы . . . . .	176
5.7	Преобразование Гивенса . . . . .	179
5.8	Варианты заполнения матрицы $R$ . . . . .	185
5.9	Правосторонние ортогональные преобразования . . . . .	186
5.10	Двусторонние ортогональные преобразования . . . . .	187
5.11	Ортогонализация Грама–Шмидта . . . . .	190
5.12	Алгоритмы ортогонализации Грама–Шмидта . . . . .	192
5.13	Решение систем после ортогонализации . . . . .	196
5.14	Обращение матриц после ортогонализации . . . . .	196
5.15	Задание на лабораторный проект № 3 . . . . .	196
5.16	Варианты задания на лабораторный проект № 3 . . . . .	198
5.17	Методические рекомендации для проекта № 3 . . . . .	199
5.18	Тестовые задачи для проекта № 3 . . . . .	208
5.19	Заключение по разделу 5 . . . . .	219

---

<b>6      Пример программной реализации проекта № 1 . . . . .</b>	<b>221</b>
6.1     Постановка задачи . . . . .	221
6.2     Класс с реализацией алгоритмов . . . . .	223
6.3     Генерация матриц . . . . .	248
6.4     Создание пользовательского интерфейса и подключение ранее созданных библиотек . . . . .	266
6.5     Завершающее тестирование . . . . .	306
6.6     Заключение по разделу 6 . . . . .	308
<b>III     ПОВЫШЕННЫЙ КУРС</b>	<b>315</b>
<b>7      Проект № 4 «Векторно-ориентированные версии           <i>LU</i>-разложения» . . . . .</b>	<b>317</b>
7.1     Гауссово исключение и $ijk$ -алгоритмы . . . . .	317
7.2     Распараллеливание вычислений . . . . .	319
7.3     Параллельное умножение матрицы на вектор . . . . .	322
7.4     Параллельное <i>LU</i> -разложение . . . . .	323
7.5 <i>LU</i> -разложение и его $ijk$ -формы . . . . .	326
7.6     Треугольные системы . . . . .	331
7.7     Задание на лабораторный проект № 4 . . . . .	333
7.8     Варианты задания на лабораторный проект № 4 . . . . .	336
7.9     Тестовые задачи для проекта № 4 . . . . .	336
7.10    Заключение по разделу 7 . . . . .	338
<b>8      Проект № 5 «Алгоритмы окаймления           в <i>LU</i>-разложении» . . . . .</b>	<b>339</b>
8.1     Метод окаймления . . . . .	339
8.2     Окаймление известной части разложения . . . . .	339
8.3     Окаймление неизвестной части разложения . . . . .	342
8.4     Задание на лабораторный проект № 5 . . . . .	344
8.5     Варианты задания на лабораторный проект № 5 . . . . .	347
8.6     Тестовые задачи для проекта № 5 . . . . .	347
8.7     Заключение по разделу 8 . . . . .	350
<b>9      Проект № 6 «Итерационные методы решения систем» . . . . .</b>	<b>351</b>
9.1     Итерационные методы . . . . .	351
9.2     Итерационная формула . . . . .	352

9.3	Метод Якоби . . . . .	353
9.4	Метод Зейделя . . . . .	353
9.5	Матричная запись методов Якоби и Зейделя . . . . .	353
9.6	Каноническая форма одношаговых ИМ . . . . .	355
9.7	Методы простой итерации, Ричардсона и Юнга . . . . .	355
9.8	Сходимость итерационных методов . . . . .	356
9.9	Скорость сходимости итерационных методов . . . . .	357
9.10	Итерационные методы вариационного типа . . . . .	360
9.11	Другие методы . . . . .	365
9.12	Задание на лабораторный проект № 6 . . . . .	366
9.13	Варианты задания на лабораторный проект № 6 . . . . .	368
9.14	Тестовые задачи для проекта № 6 . . . . .	369
9.15	Заключение по разделу 9 . . . . .	376
<b>IV</b>	<b>СПЕЦИАЛЬНЫЙ КУРС</b>	<b>377</b>
<b>10</b>	<b>Проект № 7 «Разреженные формы LU-разложения» . . . . .</b>	<b>379</b>
10.1	Упакованные формы хранения матриц . . . . .	379
10.2	Выбор ведущего элемента . . . . .	381
10.3	Задание на лабораторный проект № 7 . . . . .	384
10.4	Варианты задания на лабораторный проект № 7 . . . . .	386
10.5	Заключение по разделу 10 . . . . .	386
<b>11</b>	<b>Проект № 8 «Одновременные наименьшие квадраты» . . . . .</b>	<b>388</b>
11.1	Линейная задача наименьших квадратов . . . . .	388
11.2	Метод нормальных уравнений . . . . .	390
11.3	Формирование матрицы $A$ . . . . .	390
11.4	Задание на лабораторный проект № 8 . . . . .	391
11.5	Варианты задания на лабораторный проект № 8 . . . . .	394
11.6	Заключение по разделу 11 . . . . .	395
<b>12</b>	<b>Проект № 9 «Рекуррентные наименьшие квадраты» . . . . .</b>	<b>396</b>
12.1	Статистическая интерпретация . . . . .	396
12.2	Включение априорных статистических данных . . . . .	397
12.3	Включение предшествующего МНК-решения . . . . .	399
12.4	Рекурсия МНК в стандартной информационной форме . . . . .	401
12.5	Рекурсия МНК в стандартной ковариационной форме . . . . .	402

12.6	Ковариационный алгоритм Поттера для МНК . . . . .	406
12.7	Задание на лабораторный проект № 9 . . . . .	407
12.8	Варианты задания на лабораторный проект № 9 . . . . .	411
12.9	Заключение по разделу 12 . . . . .	414
<b>Заключение . . . . .</b>	<b>415</b>	
<b>Список иллюстраций . . . . .</b>	<b>419</b>	
<b>Список таблиц . . . . .</b>	<b>420</b>	
<b>Библиографический список . . . . .</b>	<b>421</b>	
<b>Предметный указатель . . . . .</b>	<b>425</b>	

# Предисловие

Кафедра «Информационные системы» УлГТУ решением № 7 от 24 июня 2014 года рекомендует данное учебное пособие для студентов высших учебных заведений, обучающихся по направлениям 230700 – «Прикладная информатика» (профиль «Прикладная информатика в экономике») и 321000 – «Программная инженерия». Кроме этих направлений, данное пособие рекомендуется для многих других, предусматривающих изучение студентами дисциплины под обобщающим названием «Вычислительная математика» или «Численные методы». К ним могут быть отнесены:

- 010101 – «Математика»,
- 010503 – «Математическое обеспечение и администрирование информационных систем»,
- 210400 – «Телекоммуникации»,
- 210700 – «Инфокоммуникационные технологии и системы связи»,
- 230200 – «Информационные системы»,
- 230400 – «Информационные системы и технологии»,
- 231300 – «Прикладная математика» (профиль «Математическое моделирование в экономике и технике»),
- и другие, а также
- аспирантура по специальности 05.13.18 – «Математическое моделирование, численные методы и комплексы программ».

Несмотря на различия в названиях учебных дисциплин, курс «Численные методы» по многим специальностям и направлениям подготовки в университетах преследует следующие общие цели:

- заложить базовые умения и навыки в области разработки вычислительных алгоритмов решения задач, возникающих в процессе математического моделирования законов реального мира;
- обеспечить понимание основных идей вычислительных методов, особенностей и условий их применения;
- подготовить студентов к применению этих знаний в дальнейшей учёбе и практической деятельности.

В курсе «Численные методы» значительное время отводят на изучение раздела «Вычислительная линейная алгебра». Согласно требованиям Государственного образовательного стандарта, к ВЛА относят шесть тем:

- тема 1 – методы исключения в решении систем;
- тема 2 – разложения Холесского положительно определенных матриц;
- тема 3 – методы ортогональных преобразований;
- тема 4 – итерационные методы решения систем;
- тема 5 – алгоритмы метода наименьших квадратов (МНК);
- тема 6 – методы решения проблемы собственных значений матриц.

Данное пособие покрывает пять первых тем из этого списка, при этом его отличительной особенностью является проектный метод изучения всех пяти тем. Базовым учебником для написания данного пособия послужила книга [6]: Семушин, И. В. Вычислительные методы алгебры и оценивания. – Ульяновск: УлГТУ, 2011.

Пособие структурно организовано в форме четырёх частей.

- Часть I – Основания; здесь изложены особенности нашей реализации проектно-ориентированного подхода к преподаванию и изучению дисциплины ВЛА.
- Часть II – Стандартный курс; этот материал охватывает темы 1, 2 и 3 из приведённого списка тем ВЛА.
- Часть III – Повышенный курс; здесь предлагаются две углублённые версии темы 1 из этого списка и тема 4.

- Часть IV – Специальный курс; развивает тему 1 в аспекте разреженных матриц и предлагает изучение метода наименьших квадратов в двух принципиально разных формах: (1) одновременные алгоритмы МНК и (2) последовательные (рекуррентные) алгоритмы МНК.

Значение «Численных методов» во многих областях науки и техники трудно переоценить, – и оно постоянно растёт. Важно, чтобы студенты, готовящиеся стать специалистами в области математического моделирования, численных методов и комплексов программ, обладали подлинно глубокими знаниями, т. е., знаниями, имеющими для них практическую ценность в их будущей деятельности. Такое знание достигается не схоластическим изучением теории и не решением элементарных задач в классе, но реальной проектной работой по созданию серьёзных программных продуктов высокого профессионального уровня, воплощающих эти численные методы.

В связи с этим данное пособие, как и указанный выше базовый учебник, разрабатывает так называемый проекто-ориентированный подход, при котором студенты, получив необходимый теоретический материал, закрепляют эти знания в практических лабораторных проектах. Надеемся, что при таком подходе к преподаванию и изучению студент лучше поймёт и оценит этот предмет.

Для реализации лабораторных проектов в этом пособии выбран язык C#, поскольку он является одним из основных языков программирования, изучаемых на факультете информационных систем и технологий УлГТУ, а также на факультете математики и информационных технологий УлГУ. Кроме того, язык C# применяется при разработке многих приложений в софтверных компаниях. В данном пособии студентам предоставляется шанс закрепить полученные навыки программирования и усовершенствовать их посредством разработки достаточно сложного вычислительно-ориентированного программного проекта.

Авторы искренне надеются, что данное учебное пособие поможет студентам овладеть практическими навыками реализации алгоритмов вычислительной линейной алгебры, а также навыками создания программных проектов, предназначенных для научно-исследовательских целей.

Учебное пособие разработано при частичной поддержке Российского Фонда Фундаментальных Исследований (проект № 14-07-00665).

Ульяновск  
Ноябрь 2014

Авторы

# Введение

## Учебные цели студента

Мы живём в высокотехнологичном мире, в котором компьютер уже стал неотъемлемой частью. К тому же, наше общество всё больше зависит от математики. Любая проблема решается лучше, если для неё найдена или построена подходящая (удовлетворительная, т. е., адекватная) математическая модель. При том, что для этого может потребоваться различный объём математических знаний, каждому, кто берётся решать математически ориентированные проблемы, необходимо иметь навыки аналитического мышления.

Допустим, вы этим обладаете и смогли придать задаче математическую форму, т. е., дали правильную математическую постановку задачи; вопрос заключается в том, существует ли для этой задачи аналитическое решение? Действительность такова, что множество задач, для которых аналитическое решение существует и может быть найдено в конечной форме, невелико. Большинство задач требует численных методов для своего решения. Особенность же этой области знания такова, что «наилучшего» численного метода обычно не существует, так как в одних условиях лучшим будет один метод, в то время как для других условий успешнее работает другой метод. Понять и обосновать, какой же метод выбрать как лучший, можно лишь проводя вычислительные эксперименты с различными методами и для различных задач и условий. Для этого нужно уметь осознанно планировать вычислительные эксперименты, понимать и правильно программировать численные методы и эффективно использовать возможности вычислительной техники.

Таким образом, безусловно каждому из вас потребуется хорошая компьютерная подготовка, чтобы выжить на рынке труда и успешно функционировать среди грамотных компьютерных пользователей. Было бы образовательным преступлением получить диплом выпускника университета и не иметь этих навыков хотя бы на удовлетворительном уровне. В конце концов, для этого вы и посещаете курсы информатики, программирования и численных методов.

Курс численных методов способствует этому, предоставляя преподавателю и студенту богатый набор индивидуальных заданий. В этом курсе мы преследуем три конкретные цели:

1. Студенты научатся выводить и доказывать положения математической теории численных методов, т. е., разовьют *навыки аналитического мышления*. Эти навыки будут проверены посредством *финального экзамена*.
2. Студенты увидят, как математика и компьютеры применяются к проблемам реального мира, т. е., научатся решать задачи. Эти умения будут проверены посредством *семестровых контрольных работ*, которые мы рассматриваем как часть распределённого по времени экзамена.
3. Студенты приобретут *реальный опыт* разработки компьютерных программ высокого (почти профессионального) уровня и навык применения компьютеров посредством написания, отладки и многочисленных прогонов своих программ. Приобретенный опыт будет проверен посредством выполнения *домашних заданий на лабораторные работы*, которые по своей значимости мы трактуем как весомые учебные программные проекты, имеющие большое значение.

Именно выполненные студентом проекты дадут ему возможность определить, что именно он по-настоящему изучил и понял в этом курсе.

## Оценка работы студента

**Выставление финальной оценки.** Для оценки того, в какой мере студент приблизился к своим целям — навыки, умения и опыт, — мы применяем следующую систему оценок.

- Ваша оценка есть взвешенное среднее посещаемости ( $A$ ), домашней работы ( $H$ ) и экзаменов ( $E$ ), где под «экзаменами» (см. подробнее ниже) понимается учёт не только финального экзамена (во время сессии), но и контрольных работ в течение семестра:

5 % — посещаемость.

Этот вес действует только в случае, если вы посещаете занятия. Если вы пропускаете занятия, этот вес прогрессивно снижается (см. ниже). Вы можете получить «неудовлетворительно» исключительно в результате низкой посещаемости.

**30 %** – домашняя работа.

**65 %** – экзамены.

Таким образом, итоговая оценка (final grade,  $FG$ ) вычисляется по правилу:

$$FG = 0.05A + 0.30H + 0.65E, \quad (1)$$

где каждая составляющая:

$A \equiv$  attendance (посещаемость),

$H \equiv$  homework (домашняя работа) и

$E \equiv$  exams (экзамены)

выражается целым числом не выше 100 баллов.

- Эта итоговая оценка затем отображается на стандартную шкалу оценок:

$\{82 \doteq 100\} \Rightarrow$  «отлично»,

$\{70 \doteq 81\} \Rightarrow$  «хорошо»,

$\{56 \doteq 69\} \Rightarrow$  «удовлетворительно»,

$\{0 \doteq 55\} \Rightarrow$  «неудовлетворительно».

- Пример.

Студент Иван С. имеет следующие баллы:

$A = 90$ ,  $H = 87$ ,  $E = 83$ . Тогда  $0.05 \times 90 + 0.30 \times 87 + 0.65 \times 83 = 84.6$ .

Следовательно, Иван заработал «отлично».

*Имейте в виду, что оценки зарабатываются.*

- Мы оставляем за собой право дать своего рода «плюс-минус дельта», если студент имеет оценку на границе между оценками (т. е., 81, 69 или 55). Если студент имеет 90 или выше за посещаемость ( $A \geq 90$ ), сдал все домашние задания в установленный срок и проявил хорошее прилежание, тогда мы рассматриваем возможность выставления ему следующей более высокой оценки. Если же студент не продемонстрировал указанных выше качеств, возможность повышения оценки исключается. Мы не рассматриваем возможности повышения оценки, если до граничного значения не хватает хотя бы одного балла.
- Для итоговой оценки мы используем «симметричное» округление, т. е., округляем вверх, если младшая цифра есть 5 или выше, и вниз, если она меньше пяти. При вычислении средней оценки за домашнюю работу и средней за экзамены соответствующие числа  $H$  и  $E$  округляются до ближайшей десятой, и затем они умножаются на свои весовые коэффициенты 0.05 и 0.30; после сложения по формуле (1) финальная оценка округляется.

## Учёт посещаемости (A)

- Каждое учебное занятие, в том числе лекция, начинается с вашей росписи в явочном листе. Поставить свою роспись — ваша личная ответственность. Отсутствие росписи означает ваше отсутствие на занятии. Чтобы ваше отсутствие было расценено как уважительное, вы должны известить об этом преподавателя своевременно (т. е., в течение одной недели до или после занятия). Пожалуйста, оставьте преподавателю телефонное сообщение на рабочий телефон (секретарю кафедры) или записку.
- Ваша оценка за посещаемость будет определена по табл. 1.

Таблица 1. Влияние неуважительных пропусков на оценку

Число неуважительных пропусков <sup>a</sup>	Балл <i>A</i>	Вклад в <i>FG</i> , вашу итоговую оценку
0	100	+5
1	90	+4.5
2	50	+2.5
3	0	+0
4	-50	-2.5
5	-100	-5
6	-150	-7.5
7	-200	-10
8	-400	-20
9	-600	-30
10	-800	-40

<sup>a</sup> Неуважительный пропуск есть пропуск занятия, который не связан с болезнью, с семейной утратой или с факультетским мероприятием.

При числе неуважительных пропусков выше десяти у вас нет шанса получить положительную итоговую оценку за весь курс.

- Вы можете иметь максимум 8 уважительных пропусков. После этого все пропуски считаются неуважительными.
- Для спортсмена пропуск занятия считается уважительным, если его тренер известит об этом преподавателя заранее в письменной форме. Если вы

больны, позвоните на кафедру, чтобы преподавателя об этом известили. Извещение следует делать в форме телефонного сообщения или записи секретарю кафедры. Ваше извещение должно содержать номер группы, день и время пропускаемого занятия, название предмета и, конечно, ваше имя и фамилию.

## Домашняя работа (*H*)

- Вам будет предложен ряд домашних заданий, которые — по нашему предположению — вы выполните и сдадите. Баллы за отдельные задания складываются и тем самым образуют *H*, т. е., оценку за этот вид вашей учебной работы. Любая сдача домашнего задания позже установленного срока влечёт уменьшение вашей оценки *H* на 10 баллов. За каждое невыполненное задание в *H* поступает 0.
- Домашние задания представляют собой задания на лабораторные работы (проекты). Обычно мы предлагаем выполнить 3 таких работы за семестр, т. е., выдаём 3 задания. Максимальное количество баллов *H*, которое можно заработать за всю домашнюю работу, составляет 100. Эти 100 баллов мы разделяем определённым образом между общим числом выданных домашних заданий.
- Предлагаемые каждому студенту 3 лабораторные работы на первый семестр покрывают три темы из списка на стр. 15, включённые в данное пособие. За выполненный безупречно и в полном объёме проект № 1 студент заработает 50 баллов, причём по срокам это задание должно предшествовать всем последующим. Далее, за выполненное безупречно и в полном объёме проект № 2 студент заработает 20 баллов, а за выполненное безупречно и в полном объёме задание проект № 3 – 30 баллов. Заработанное число баллов за каждое задание будет уменьшено, если защита работы не отвечает всем требованиям, изложенным в данном учебном пособии, или не демонстрирует самостоятельность выполнения.
- Обнаруженная несамостоятельность выполнения (плагиат, т. е., выдача чужого проекта за свой) имеет своим результатом то, что в *H* немедленно поступает 0.
- Следующие по номерам проекты могут предлагаться в рамках дополнительного семестра или как курсовые работы.
- Преподаватель, ведущий лабораторные занятия в дисплейном классе, назначит сроки сдачи лабораторных работ и на каждом занятии всегда с

готовностью поможет вам, если вы ясно формулируете те конкретные вопросы, которые у вас возникли. Преподаватель, ведущий семинарские (практические) занятия, поможет вам и всей аудитории, когда вы будете у доски рассказывать, как вы понимаете и как дома программируете тот или иной алгоритм.

## Экзамены ( $E$ )

- Ваша оценка за экзамены, т. е., величина  $E$  в составе финальной оценки, вычисляемой по формуле (1), будет определена как равномерно взвешенное среднее значение результатов  $P_{KP-1}$ ,  $P_{KP-2}$  и  $P_{KP-3}$  трёх письменных контрольных работ ( $KP-1$ ), ( $KP-2$ ) и ( $KP-3$ ) в течение семестра и результата  $P_{yo}$  устного ответа ( $YO$ ) на зачёте (во время зачётной недели) или на экзамене (во время экзаменационной сессии). Это означает, что

$$E = (P_{KP-1} + P_{KP-2} + P_{KP-3} + P_{yo}) / 4, \quad P_{KP-i}, P_{yo} \in [0, 100]. \quad (2)$$

При том, что контрольные работы письменно проверяют ваше умение решать задачи, устный зачёт или экзамен есть всеобъемлющая проверка вашего знания основных положений теории, умения доказывать эти положения и делать из них логические выводы. Эти (письменная и устная) части  $E$  (2) в совокупности покрывают весь учебный курс. Для этого мы проводим обычно три контрольные работы за семестр.

- Все контрольные работы будут вам объявлены заранее – не позднее, чем за неделю. Если вы собираетесь пропустить контрольную работу (это должен быть уважительный пропуск), мы предпочтём, чтобы вы написали эту работу раньше назначенного срока. Если вы не сможете написать контрольную работу до назначенного срока, то примите все меры к тому, чтобы написать её в течение недели после контрольного срока. По истечении недели после этого вы получите ноль. Вы также получите ноль за неуважительный пропуск контрольной работы.
- Мы переписываем и заменяем некоторые задания или делаем вариации в постановке экзаменационных вопросов по сравнению с теми, которые опубликованы в наших учебных пособиях, в предыдущем семестре или в типовой рабочей программе на нашем сайте. Об этом будет объявлено за две недели до контрольных работ или финального экзамена.

# Кодекс студента

## Академическая честность

- К сожалению, есть люди, не столь честные, как другие, и настолько, что мы должны пояснить, как будем действовать в этом случае.
- За любую контрольную работу, экзамен, программу или любой иной вид работы, который выполнен нечестно, вы получите ноль, и преподаватель будет беседовать с вами. Если такая проблема случится во второй раз, преподаватель направит вас к декану факультета, и вы снова заработаете ноль за этот вид работы. Если вопрос о нечестности возникнет в третий раз, то вы сразу заработаете «неудовлетворительно» за весь предмет и снова будете отправлены к декану.
- Что считается *академической нечестностью*, т. е., обманом? По общепринятым правилам, это означает найти кого-то другого, кто сделает за вас вашу работу, и выдать её за вашу собственную. Это также включает получение и оказание посторонней помощи на экзамене или во время контрольной работы (от соседа или с помощью шпаргалки).
- *Наши экзамены — это всегда закрытая книга, закрытый конспект, закрытый сосед и открытый ум.* Если в этом правиле появятся какие-либо изменения, об этом будет объявлено заранее.
- Не пользуйтесь шпаргалками. Они приносят больше вреда, чем пользы. Ваше сознание будет раздвоено между попыткой сформулировать ответ и попыткой утаить факт пользования шпаргалкой. Обнаружить такое раздвоенное сознание не составляет никакого труда. Вы будете обескуражены ещё больше самыми простыми вопросами экзаменатора.
- При выполнении домашних заданий приемлемо работать с кем-то ещё, обсуждая трудные вопросы и помогая тем самым друг другу, но при этом вы должны сами делать свою работу. Например, при написании компьютерных программ вполне нормально — обсуждать синтаксис, детали задания или получать помощь по сообщениям об ошибке. Ненормально, если вы отдаёте кому-то копию вашей программы. Неприемлемо, если кто-то другой пишет программу для вас. Недопустимо копировать работу предыдущего семестра.
- В курсовых работах — вообще, в любых письменных работах — *плагиатом* является дословное копирование части чужих трудов, таких как чья-то

статья (в том числе и опубликованная в Интернете), книга или энциклопедия, без использования кавычек и ссылки на источник. Обобщающие заключения и выводы, которые вы пишете, должны быть выражены вашими собственными словами.

- Нечестность, когда она появляется в домашней работе, не столь очевидна. Мы это вполне признаём. Но она так или иначе проявит себя на устном зачёте или экзамене, так как ваш балл за домашнюю работу будет контрастировать с уровнем вашего ответа. Вы только навредите себе и осложните свое положение очевидной провинностью.

## **Поведение в аудитории**

- Примите все меры к тому, чтобы приходить на занятия вовремя. Если вы всё же опаздываете,
  - не спрашивайте разрешения войти и не извиняйтесь за опоздание,
  - не проходите на место перед передним рядом мест,
  - тихо займите ваше место,
  - для получения любого раздаточного материала (если он есть) дождитесь конца занятия,
  - не хлопайте дверью.
- Чтобы выйти из аудитории, просите разрешения.
- Поднимайте руку и ждите, когда на вас обратят внимание, перед тем как задать вопрос.
- Не разговаривайте в аудитории.
- Уберите за собой и поставьте стул в исходное положение.

## **Путь к успеху**

- Приходите на занятие вовремя, принимайте в нём участие и ведите записи.
- Просматривайте задания до занятия.
- Проверяйте ваши записи после занятия.
- Вовремя выполняйте ваши задания.
- Не накапливайте задолженности по самостоятельной работе – чтению учебных материалов, по домашней работе и в целом – по учебе.

- Выполняйте рекомендации по подготовке к контрольным работам и к финальному экзамену. Убедитесь, что вы можете решать типовые задачи и доказывать теоремы, которые во время лекций были отмечены как упражнения для самостоятельной работы.
- Придерживайтесь твёрдой решимости добиться успеха.
- Если вам нужна помощь, получайте её безотлагательно.
- Сохраняйте позитивное отношение к делу.

## Обратная связь

- По окончании всего курса занятий заполните анонимно лист обратной связи, который выдаст преподаватель. В нём вы можете отметить как положительные, так и отрицательные, на ваш взгляд, стороны преподавания этого предмета в текущем семестре.
- Преподаватель периодически просматривает свой сайт. Вы можете посыпать через него ваши кратко сформулированные предложения, которые, по вашему мнению, помогли бы вам повысить эффективность изучения этого предмета.

*Добро пожаловать на наши занятия. Вместе мы рассчитываем на большой и продуктивный семестр!*

## Краткое описание курса

Показанный ниже примерный план (с. 26) может незначительно варьироваться в соответствии с новыми образовательными стандартами. Однако как бы ни называлась дисциплина, соответствующая этому плану, — «Численные методы», «Вычислительная математика» или «Методы вычислений», — главная отличительная особенность нашего курса и данного пособия выражается в следующем диалоге, который иногда возникает между Студентом и Экзаменатором:

Студент: Я знаю этот численный метод и хочу получить более высокую оценку.

Экзаменатор: Если вы хорошо знаете этот метод, тогда научите этому компьютер.

<b>ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ (АЛГЕБРЫ)</b>						
<b>Кредиты:</b> 3 <sup>a</sup>	<b>Семестр:</b> 1 <sup>b</sup>	<b>Обязательный:</b> да	<b>Экзамен</b>			
<b>Формат:</b>	Лекции Семинары Лабораторные работы Самостоятельная работа	34 ч 0 ч 17 ч 49 ч				
<b>Преподаватель:</b> Будет объявлен за четыре месяца до начала семестра.						
<b>Содержание:</b>						
Методы решения базовых задач вычислительной линейной алгебры: решение систем уравнений, обращение матриц и вычисление определителей. Программа курса охватывает: методы исключения неизвестных (факторизация матриц), разложения Холесского, ортогональные преобразования, итерационные методы, задачи ВЛА с разреженными матрицами, метод наименьших квадратов и его численно устойчивые алгоритмы.						
<b>Ожидаемые результаты изучения:</b>						
<b>знати и понимать:</b>	вопросы погрешностей вычислительных методов; обусловленность задач и методов; прямые и итерационные методы решения систем; особенности вычислений с разреженными матрицами; задачи и алгоритмы метода наименьших квадратов.					
<b>способность:</b> (теоретические навыки)	понимать и формулировать основные численные процедуры ВЛА; решать демонстрационные задачи; идентифицировать численные методы для конкретных задач линейной алгебры.					
<b>способность:</b> (практические навыки)	понимать реализацию и поведение численных методов; логически формулировать алгоритмы ВЛА для программирования на языках высокого уровня (C #, Pascal или C/C++).					
<b>способность:</b> (ключевые навыки)	осваивать предмет самостоятельно; анализировать литературные источники; профессионально использовать компьютер; планировать работу и эффективно распоряжаться своим временем.					
<b>Оценивание:</b> 5 % за посещаемость (неуважительные пропуски прогрессивно штрафуются); 30 % за семестровые (домашние) задания, 65 % суммарно за три контрольные работы и финальный (устный) экзамен.						
<b>Семестровые (домашние) задания:</b> Студент разрабатывает программные проекты, реализующие численные методы решения базовых задач ВЛА и позволяющие проводить широкие вычислительные эксперименты.						
<b>Рекомендуемые учебные материалы:</b>						
1. И. В. Семушин. Вычислительные методы алгебры и оценивания. – Ульяновск: УлГТУ, 2011. [Доступ: <a href="http://venec.ulstu.ru/lib/disk/2013/119.pdf">http://venec.ulstu.ru/lib/disk/2013/119.pdf</a> ]						
2. В. В. Воеводин. Численные методы алгебры. Теория и алгоритмы. – М.: Наука, 1966.						
3. А. А. Самарский, А. В. Гулин. Численные методы. – М.: Наука, 1989.						
<b>Дополнительное чтение:</b>						
4. Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков. Численные методы. – М.: Наука, 1987.						
5. Дж. Райс. Матричные вычисления и математическое обеспечение – М.: Мир, 1984.						

<sup>a</sup> Число кредитных часов (приравнивается числу аудиторных часов в неделю).

<sup>b</sup> Продолжительность курса.

# I

# ОСНОВАНИЯ



# 1

## Проекто-ориентированная методика

### 1.1 Метод проектов

#### Учиться, делая

Развитие навыков глубокого мышления является одной из главных забот педагога. Исследования показывают, что использование компьютера как инструмента познания открывает большие возможности [47]. Однако ещё большие возможности заключает в себе разработка программ для компьютера. Здесь студент реализует и ощущает себя не как простой пользователь готового, но как создатель нового. Разработка, проектирование – это творческая деятельность, рассчитанная на пытливость молодого ума. Включение психологических механизмов, мотивирующих к такой деятельности, меняет всю «философию» педагога. Образовательное взаимодействие преподавателя и студента из «препо-центрированного» превращается в «студент-центрированный» процесс. Фокус процесса смещается от «преподавателя-рассказчика» к студенту, который «учится, делая» [58]. Об этом говорил ещё Аристотель, – см. [6]: «Ибо то, что нам надо постигать, чтобы уметь это делать, мы постигаем, делая это».

#### Проекто-ориентированное обучение: обоснования и преимущества

Проектный метод имеет долгую традицию в науке и практике образования. Так, в [49] читаем:

«Kilpatrick [48] отстаивал «проектный метод», а Dewey [44] продвигал «обучение из опыта». В типичном случае обучение, основанное на проектах, имеет пять характеристик: (а) целенаправленность, (б) ведущий вопрос, (в) подлинность (соответствие реальности), (г) конструктивное исследование и (д) самостоятельность студента [58].

В проекто-ориентированном образовании (ПОО) проекту обычно отводится роль главной стратегии преподавания, которая побуждает студентов изучать и самостоятельно наталкиваться на то, что им необходимо изучить. На практике ПОО обычно стартует с одного ведущего вопроса, вынуждающего студентов осваивать центральные понятия и принципы некоторой темы. Этот ведущий вопрос, как правило, тесно связан с тем, что встречается в реальной жизни, при этом от студентов требуется исполнение реальных ролей и решение вполне осмысленных задач. Чтобы найти ответ на поставленный вопрос, студенты вовлекаются в познавательные процессы: решение задач, принятие решений, проектирование и вдумчивое размышление. Это помогает студентам преобразовывать информацию и конструировать своё собственное знание и толкование. Такой процесс поощряет студентов брать ответственность за своё обучение и становиться самостоятельными в своих решениях. В этом процессе к студентам приходит высокое чувство собственного достижения».

ПОО широко распространено за рубежом и постепенно находит применение в Российских образовательных учреждениях. Оно реализуется в различных формах и способах.

## Всемирная инициатива CDIO

В настоящее время наиболее заметной формой и способом реализации ПОО является *Всемирная инициатива CDIO* (*Conceive ⇒ Design ⇒ Implement ⇒ Operate*) [32], [43]. Её замысел, возникший в 1999 году, был продиктован потребностями нашего высокотехнологичного времени и документально оформлен в 2000 году в трёх университетах Швеции: Royal Institute of Technology in Stockholm, Linköping University in Linköping, Chalmers University of Technology in Göteborg, и в ведущем техническом университете США – Massachusetts Institute of Technology (MIT).

При поддержке фонда Wallenberg Foundation эти университеты провозгласили программу международного сотрудничества, разработанную ими для улучшения додипломного инженерного образования в Швеции, в США и по всему миру. Сначала эта тесно скоординированная программа объединяла усилия четырёх университетов-инициаторов программы. С тех пор число её участников постоянно растёт.

Общий замысел этого проекта следующий: погрузить студентов в такой образовательный процесс, в котором фундаментальные дисциплины инже-

нерного знания осваиваются студентами в контексте «Задумай ⇒ Спроектируй ⇒ Реализуй ⇒ Управляй» (CDIO), типичном для систем и изделий реального мира.

На конец 2014 года в программе-консорциуме CDIO участвуют 114 вузов мира в семи регионах (Европа, Северная Америка, Азия, Латинская Америка, Великобритания–Ирландия, Австралия–Новая Зеландия и Африка). В CDIO участвуют пока только девять университетов России:

1. Астраханский государственный технический университет (с 2012)
2. Московский авиационный институт (с 2012)
3. Московский физико-технический институт (с 2013)
4. Национальный университет ядерных исследований (с 2014)
5. Сибирский федеральный университет (с 2012)
6. Сколковский научно-технический институт (с 2012)
7. Томский политехнический институт (с 2011)
8. Томский технический институт систем управления и радиоэлектроники (с 2013)
9. Уральский федеральный университет (с 2013)

Расширенное заседание Правительства РФ 31.01.2013 поставило задачи поддержки этого направления в вузах России ввиду острого дефицита инженеров, полагая, что это поможет решить задачи опережающей подготовки инженеров для инновационного технологического развития страны.

Доводы для Российских университетов в пользу участия в CDIO:

1. Государственная поддержка.
2. Конкурентное преимущество при наборе абитуриентов.
3. Влияние на рейтинг вуза в системе Минобрнауки.
4. Доступ к базе данных коммерческих и государственных заказов на высокотехнологические разработки.
5. Участие в распределении целевых грантов и субсидий на развитие и достройку своей инфраструктуры под требования стандартов.

6. Доступ к участию в международных конкурсах и чемпионатах управлена-  
ческих компетенций и надпредметных навыков.
7. Другое (общая престижность и значимость) ...

В Ульяновске проектно-ориентированное обучение возникло в 1989 году, когда под эгидой Московского государственного университета им М. В. Ломоносова зарождался Ульяновский государственный университет и в составе нового университета начали функционировать механико-математический факультет и кафедра Математической кибернетики и информатики. В 1999 году от кафедры МКИ для доклада на ICME-9 (9-й Международный конгресс по математическому образованию) появился материал [52] по фронтально-состязательному подходу к преподаванию вычислительной математики (ВМ). В 2002 и 2003 годах были сделаны доклады о проектно-ориентированном обучении в области вычислительной математики и информатики (ВМИ) [54], [55], [56] на трёх международных конференциях.

## 1.2 Фронтально-состязательный подход (ФСП)

### Теории обучения

Большинство дискуссий в теориях обучения отмечают важность мотивации, обратной связи и поощрения [45]. Замечено, что компьютеры улучшают мотивацию к обучению, особенно для людей с физическими ограничениями или же одарённых, тем, что они: (1) занимают внимание, (2) индивидуализируют преподавание, (3) обеспечивают доступ к поучительным ситуациям и впечатлениям, что трудно обеспечить иными средствами, (4) предоставляют возможности и средства коммуникации. По Malone (1981), как отмечается в [45], истинная мотивация означает широкие возможности для выбора сложности решаемых задач, обратной связи и чётко очерченных критериев качества учебной работы.

«Обратная связь и поощрение – два ключевых момента в обучении.

Обратная связь снабжает учащегося информацией о его ответах, в то время как поощрение усиливает тенденцию к повторению некоторого специфического ответа. Обратная связь может быть положительной, отрицательной или нейтральной. Информационная теория обучения придаёт большее значение обратной связи, в то время как бихевиористская теория акцентирует поощрение. В обоих случаях одной из

критических величин является промежуток времени между ответом и обратной связью или поощрением, и чем меньше этот промежуток, тем легче и естественнее протекает процесс обучения».

Однако вести учебный процесс в точном соответствии с принципом «стимул-реакция-обратная связь/поощрение» чрезвычайно трудоёмко, но со взрослыми людьми ещё и непросто. Преподаватели университета тем и отличаются от учителей школы, что они не проводят каждый день «уроки» или контрольные работы или зачёты и не вступают в существенные взаимообмены информацией, пока студенты сами не обращаются с вопросами. Они, скорее, инструктируют студентов и дают им волю «впитывать» содержание курса посредством значительной по объёму самостоятельной работы. Справедливо говорит Richard Dorf, «То, что мы по-настоящему изучили и поняли, мы открывали сами» [17]. Это снова убеждает нас в том, что *проекто-ориентированная методика* соединяет все преимущества:

- *Целенаправленность.* Задание на проект (работу) должно настраивать студента или группу студентов, если проект групповой, на достижение одной цели, которую они сами способны понять и сформулировать.
- *Подлинность.* Задание должно соответствовать реальности, то есть формулировать задачи, соответствующие действительности.
- *Вызов.* Задание должно предусматривать возрастающие уровни сложности, бросающие студенту вызов с тем, чтобы актуализировать (приводить в действие) все возможности личности, её творческий потенциал и состязательный инстинкт.
- *Разнообразие.* Набор заданий должен создавать возможность свободного выбора несовпадающих тем и предусматривать различные сценарии выполнения для поддержания интереса.
- *Поощрение.* Оценивание достигнутого студентом уровня должно быть количественно измеримо и понятно студенту по мере улучшения приобретённых им навыков и согласно текущей общей успешности выполнения задания (распределённое по времени градуированное поощрение).
- *Навигация.* Студент должен иметь возможность самостоятельно осуществлять навигацию по любым сценариям выполнения проекта (контролируемая студентом навигация) для получения желаемой оценки и – в конечном итоге – для достижения своих личных образовательных целей.

## Особенности математики и информатики

Математика и информатика как предметы для преподавания или изучения обладают той особенностью, что не могут быть просто «вложены» в память учащегося. Это означает, что хороший студент не хочет останавливаться на первом уровне знания, т. е., на уровне простого понимания. Он стремится перейти на следующий уровень, – уровень самостоятельного воспроизведения того, что понял. Хотя и это уже неплохо, некоторые студенты не останавливаются на этом. Для них важен третий уровень, который означает способность усомниться в качестве имеющегося решения, вывести новый результат и обосновать его преимущества.

Чтобы подняться с первого уровня, студент пропускает через себя всю информацию посредством самостоятельного решения большого числа задач. Однако этого недостаточно. Как говорил венгерский математик Alfréd Rényi, «Тот, кто изучает решение без понимания сути, не может использовать его надлежащим образом» [35]. Независимый, критический подход и творчество – это те качества, которые нужны студенту для достижения третьего уровня, и только такое знание имеет реальную ценность, когда изучают математику или информатику.

Другое свойство математики заключается в том, что её часто называют «кабинетной» наукой. Однако для многих университетов характерны большие аудитории студентов до такой степени, что их обозначают термином «поток».

Почти очевидно, что большой поток студентов служит своеобразным препятствием на пути развития независимого, самостоятельного мышления. Многие студенты предпочитают «плыть по течению», нежели трудиться независимо, т. е., хотят быть «как все». Как превратить это препятствие в преимущество? Каким образом преподаватель может поощрить студенческую независимость? Как он может помочь студентам понять их выдающиеся способности? И, наконец, как мы можем доказать им, что математика – это не собрание неопровергаемых и малопонятных фактов, а живая, красивая наука, дающая жизнь другим важным наукам, в частности, информатике?

К счастью, математика сама по себе подсказывает нам, как достичь этих целей. Мы демонстрируем это в данном пособии на примере Вычислительной математики и представляем ниже *Фронтально-состязательный подход* к её преподаванию.

## Главная идея ФСП

Она может быть объяснена самим её названием. «Фронтальный» означает вовлекающий всю аудиторию в достижение одной цели. «Состязательный» означает дающий возможность достижения успеха, благодаря индивидуальным творческим и нестандартным решениям или действиям. Чтобы воплотить ФСП в жизнь, мы делаем следующее:

1. Организуем творческую обстановку.
2. Пробуждаем творческий потенциал студента.
3. Даём толчок студенческому инстинкту соперничества.
4. Обеспечиваем прозрачность критериев оценивания.

Рассмотрим эти компоненты ФСА чуть детальнее в приложении к дисциплине Вычислительная математика.

### Творческая обстановка

Большинство существующих учебных материалов по ВМ сообщают основные теоретические сведения и некоторые теоретические инструкции, как за-программировать численный метод или алгоритм. Однако это не вполне соответствует конечной цели. Мы считаем, что истинное понимание численного метода достигается, если: (а) студент выполняет сложное проектное задание по разработке программного продукта, реализующего заданный численный метод; (б) студент проводит серию вычислительных экспериментов с этим программным продуктом и, наконец, (в) фронтальное оценивание проектов выполняется коллективно студентами и преподавателем.

Программирование само по себе даёт несколько выгод студенту. Во-первых, оно позволяет студенту понять и изучить численный метод «изнутри». Для творческой личности, для растущего профессионала это совсем не то, что использовать готовый программный продукт. Во-вторых, это улучшает компьютерную грамотность студента, так как требует изощрённого программирования. И, наконец, это развивает общие способности аналитического мышления, навык поиска решений, прививает умение без боязни браться за решение математически ориентированных задач, которыми так полон наш мир.

Организовать творческую обстановку в процессе преподавания ВМ в больших аудиториях можно лишь при большом разнообразии учебных заданий по программированию вычислительных методов. Важно при этом, что задания

должны различаться по алгоритмам, а не просто по исходным данным для одного и того же метода. Поскольку число вариаций численного метода обычно ограничено, а количество студентов велико, отыскание как можно большего разнообразия версий численного метода становится вопросом большой методологической важности для каждого преподавателя.

Организация творческой обстановки означает также, что мы должны оценивать каждый лабораторный проект по программированию как уникальный продукт, который обладает всеми качествами завершённости, а именно, имеет: (а) выраженную модульную структуру, (б) удобный интерфейс, (в) эффективное использование вычислительных ресурсов (памяти и времени) и (г) возможности проводить широкий вычислительный эксперимент. Это решительным образом отличается от широко распространённой практики, когда студенты работают на одном и том же готовом программном продукте как его пользователи, когда они только вводят свои исходные данные и пассивно ждут ответа. Подход, который мы практикуем, вынуждает студентов совершать творческие действия, он в каждом студенте стимулирует врождённый дух соперничества и, в целом, помогает улучшить качество работы всех, кто в этом потоке. Кроме того, эта обстановка предотвращает нечестность, списывание, поскольку поощряет студента к превосходству над другими.

Классическим примером того, как можно разнообразить задания по численным методам, может служить тема «Методы исключения и обращение матриц». Сначала преподаватель систематизирует алгоритмы Гаусса или Гаусса-Жордана по их особым характеристикам. Среди них могут быть: (1) направление нумерации исключаемых неизвестных, (2) направление перебора уравнений в процессе исключения, (3) способ доступа к элементам матриц, (4) режим обновления активной подматрицы, (5) стратегия выбора ведущего элемента. Затем, учитывая взаимную независимость этих характеристик, преподаватель комбинирует их так, что в результате получает значительное число вариантов заданий по этой теме.

### Творческий потенциал студента

Лекции обычно включают доказательство одной версии численного метода, например, теорему об  $LU$ -факторизации матриц, теорему об  $LU$ -факторизации матриц с выбором ведущего элемента и теорему об алгоритме  $LU$ -факторизации с замещением исходной матрицы факторами  $L$  и  $U$ . Однако практически каждая из таких теорем имеет своего «двойника»; в нашем примере,  $UL$ -,  $UDL$ - или  $LDU$ -разложения также возможны. То же самое отно-

сится и к выбору ведущего элемента: возможны, по крайней мере, три стратегии его выбора.

Следовательно, будет целесообразно, если студент, выполняющий задание на проект по этой теме, самостоятельно докажет теоремы по своей версии численного метода. Действуя таким образом, студент учится не только доказывать (что само по себе очень важно в математическом образовании), но привыкает смотреть на математические вещи шире. Его творческие способности возрастают, потенциал превращается в активность, и это достижение может быть специальным образом отмечено, особенно, в отношении одарённых студентов.

### Инстинкт соперничества

Как правило, студенты математических подразделений полны желания быть профессионалами в компьютерах и современных технологиях программирования. Им неинтересно «топтаться» на начальном уровне компьютерной грамотности. Они определённо выражают желание показать свои навыки в создании «выдающихся» программ. ФСП идеально поддерживает этот инстинкт соперничества. Действительно, множественность вариантов заданий по одной и той же теме удачно сочетает две стороны: общность и различие вариантов. Благодаря наличию общих черт в заданиях, студенческие проекты могут быть сравнены, а благодаря различиям в особенностях, проекты становятся индивидуальны по природе.

Преподаватели, практикующие ФСП, наблюдали замечательные случаи, когда студент, уже получивший зачёт по проекту с разреженными матрицами, продолжал обновлять свою программу, изменяя схему доступа к элементам матриц, чтобы добиться более высокого быстродействия, потому что у его коллеги-студента быстродействие программы было выше. Иногда студенты устраивают своего рода соревнование, чья программа имеет более удобный интерфейс или быстрее работает. Иногда в период каникул мы устраиваем презентации лучших студенческих проектов. На первых порах мы требовали, чтобы программы были разработаны на языке Pascal, но сейчас мы допускаем использование разных языков и инструментов программирования: Visual Basic, Delphi, Builder C++, C#, MATLAB, Scilab и др.

### Прозрачность оценивания

Роль преподавателя в рамках ФСП очень велика. Кроме отмеченного выше, преподаватель должен выдвинуть точную и определённую критериальную систему требований и систему выставления оценок за работу студентов над про-

ектами. Любой студент должен с полной определённостью знать, какую оценку и за какое количество и качество работы он получит. Приступая к работе, студент сам выбирает тот уровень оценки, на который он претендует. Система оценивания должна быть так разработана, чтобы студент имел возможность самостоятельно перемещаться на другой уровень, если первоначальный уровень претензий оказался занижен или завышен.

Например, в нашем проекто-ориентированном курсе «Практикум по методам оптимизации» [38]–[39] система заданий по линейному программированию насчитывает 70 вариантов проекта при том, что число базовых алгоритмов симплекс-метода невелико: стандартный, двойственный или модифицированный алгоритм. Однако все задания разделены на три группы в зависимости от их сложности: базовый уровень (20 вариантов), продвинутый уровень (30 вариантов) и высокий уровень (20 вариантов). Оценки за них назначены соответственно: «удовлетворительно», «хорошо», «отлично». Такая ясность критериев и возможность независимой «навигации» студента по соответствующим уровням сложности проекта оказывают заметное влияние на студенческую активность.

## **ФСП + ПОО = изменение поведения [53]**

Строго говоря, ФСП не является нашим оригинальным изобретением. Он применяется, – хотя так специально и не формулируется, – преподавателями различных университетов на постсоветском пространстве и, наверное, применялся ранее.

Наша работа в Филиале МГУ им. М. В. Ломоносова в г. Ульяновске (1988–1996) и далее в Ульяновском государственном университете и Ульяновском государственном техническом университете на протяжении многих лет фокусировалась на возможных способах приложения ФСП к преподаванию численных методов в курсах линейной алгебры, рекуррентных наименьших квадратов, оптимальной фильтрации, оптимального управления, линейного программирования, исследования операций, языков программирования и других дисциплин. Применение ФСП в сочетании с ПОО доказало, что студенты в своём большинстве дают положительный отклик на эту методику. Индивидуальная работа в составе большой аудитории побуждает студентов к жёсткой, но оправданной состязательности, к поиску творческих решений и к лучшим оценкам качества.

Однако то, что изложено выше, есть лишь дидактическая предпосылка, отвечающая на вопрос: ЧТО должно быть сделано для того, чтобы развить в студентах «вычислительные» таланты и универсальные навыки профессионала

в области вычислительных наук или прикладной математики и информатики. Формой практического воплощения ФСП, позволяющей получить желаемый результат, мы считаем ПОО – *проекто-ориентированное обучение*, и тем самым отвечаем на вопрос: КАК это делать, чтобы изменить поведение студентов к лучшему.

## 1.3 Оценивание качества академических проектов

Очевидно, внедрение новых методик в сфере инженерного образования имеет своей конечной целью повышение качества обучения. Важнейшей проблемой при этом является оценивание эффективности внедряемой методики: Насколько она полезна? Сильно ли она влияет на достижение этой конечной цели? Определить это можно лишь в ходе педагогического эксперимента, например, методом экспертных заключений. Однако это – не единственный возможный метод.

В данном разделе излагается *методика оценивания качества программных проектов* студентов, в которой качество продукта определяется количественно компьютерной программой без участия экспертной группы.

### Главные проблемы качества образования

Современное производство быстро усложняется. Появление всё более сложных изделий требует от сферы образования соответствующего повышения уровня подготовки выпускников. В этом деле подготовки современных специалистов для высшей школы обнажились несколько проблем, которые надо рассматривать как ключевые, критические.

#### Проблема 1. Мотивация студентов

Пониженную мотивацию студентов к обучению замечают многие и замечают не только в России. Многим известно обращение Королевского математического общества и Общества статистиков Великобритании к правительству страны по поводу снижающегося качества математического образования [51] и предложения правительству о том, как справиться с этой проблемой [57]. Россия – не исключение. Пониженная мотивация наших студентов к обучению обусловлена множеством причин и просчётов системы образования. Их рассмотрение выходит за рамки данного текста. Главное для нас здесь – не поиск причин, а конструктивное предложение выхода из негативной ситуации.

Как уже отмечено (см. разд. 1.1), полезным решением является проектно-ориентированное обучение. Фронтально-состязательный подход (см. разд. 1.2) может этому способствовать.

В своей работе [34] J. Christopher Jones привёл некоторые рекомендации преподавателям, применяющих проектно-ориентированный подход:

1. Давать возможность самим студентам выявлять проблему и цель.
2. Разработать такие задачи, в которых это возможно и поощряется.
3. Не критиковать результаты, а подвергать сомнению сам процесс.
4. Предлагать проекты как на ближайшую, так и на отдалённую перспективу.
5. Показывать, как вы сами проектируете.
6. Демонстрировать новые методы в позиции не эксперта, а обучающегося.
7. Предлагать студентам отложить работу, которая их не привлекает.
8. Учить студентов быть готовыми начать процесс проектирования заново.

Совместная проектная деятельность (в групповых проектах) позволяет развивать творческое начало в деятельности студента. Проектный метод в обучении как активная форма организации учебного процесса позволяет студентам развивать свои способности, приобретать знания, умения и навыки для решения практических, жизненно важных задач.

Проектное обучение студентов развивает в них особенные качества:

1. Исследовательские умения, т. е., способность анализировать проблемную ситуацию, работать с литературными источниками, формулировать решаемую задачу, проводить наблюдение практических ситуаций, строить гипотезы и делать выводы.
2. Умение работать в команде, осознавать значимость коллективной работы для получения результата, ценить роль сотрудничества и взаимопомощи.
3. Коммуникативные навыки, т. е., привычка формулировать и высказывать свою точку зрения, выслушивать и понимать другие мнения, критически подходить к своим и чужим суждениям в процессе поиска решения задачи.

### Проблема 2. Связь учебных заданий с производством

В реальных условиях работы наших учебных заведений учебные проекты, предлагаемые при реализации проекто-ориентированного обучения, – особенно на его начальной стадии, – бывают далеки от тематики проектов, которыми занимаются профессиональные инженеры в рамках своей производственной деятельности. J. Michael Fitzpatrick и Ákos Lédeczi пишут [46]:

«В университетах содержание курсов информационных дисциплин диктовалось потребностями крупных софтверных фирм. Однако инженеры и программисты подходят к освоению методов программирования с разными целями. В то время как инженеры создают программы, в основном, для собственного использования, программисты создают программы для использования непрограммистами».

Проекты по дисциплине Программирование, реализуемые в учебном процессе, серьёзно отличаются от профессионального проектирования программного обеспечения. Это обусловлено тем, что профессиональные продукты предназначены для вывода на рынок, и это обстоятельство создаёт сильную обратную связь по качеству продукта и по его соответствию требованиям потребителей. В то же время проекты, выполняемые в учебном процессе, не имеют непосредственного контакта с рынком (связь очень слабая). Программы, созданные студентами во время учебных занятий, мы называем *академическими программными продуктами*, сокращённо – АПП. Жизненный цикл АПП заканчивается на этапе аттестации, отсутствуют этапы эксплуатации и сопровождения, очень мала доля тестирования. Поэтому при разработке тематики проектов для обучения необходимо повышать их практическую значимость. Этапы разработки в большей степени должны следовать современным технологиям разработки проектов.

### Проблема 3. Оценивание качества проектных решений

Дополнительной проблемой при внедрении проекто-ориентированного подхода является получение адекватной (правильной) оценки качества предлагаемых студентами проектных решений. При традиционном подходе оценка качества АПП формулируется экспертами, в качестве которых выступают преподаватели. Недостатком такого метода является повышенная трудоёмкость анализа учебного программного продукта. Свои представления о требованиях к знаниям, навыкам и умениям студентов преподаватели вырабатывали на протяжении длительного времени, опираясь на свой опыт.

Однако требования преподавателей, несмотря на опыт и квалификацию, существуют на интуитивном уровне, и они сугубо индивидуальны. Высокий разброс требований в вузе к навыкам и знаниям студентов приводит к тому, что выпускники, придя на производство, сталкиваются с серьёзным расхождением представлений о процессах проектирования в учебном заведении и на производстве. Это неизбежно приводит к необходимости переучиваться и изменять сложившийся стереотип.

### **Возможности оценивания АПП**

Академический программный продукт существует в виде файла с размещённым в нём текстом программы на языке высокого уровня. Будучи конечным продуктом реализации проекта, он обладает определёнными различиями по сравнению с другими формами отчётности:

1. Этот продукт изначально представлен в электронном виде и готов к обработке на компьютере.
2. Его логическая структура с точки зрения грамматики и синтаксиса не имеет ошибок, поскольку они обнаруживаются компилятором на этапе отладки.
3. Его работоспособность и соответствие учебному заданию доказана экспертом, в роли которого выступает преподаватель, а в рамках проектно-ориентированного обучения экспертами выступают все члены группы, реализующие данный проект.

Это свидетельствует о том, что АПП может быть использован в процессе формирования оценки качества проекта. Оценка может быть получена в автоматическом режиме, независимом от экспертов. Такой подход позволяет обрабатывать значительные объёмы академических программных продуктов, давать им независимую, объективную оценку и даже выполнять сравнительный анализ отдельных групп, отдельных «команд» студентов, а также методов обучения.

### **Инструментарий проектно-ориентированного обучения**

В ходе учебного процесса студентами кафедр «Информационные системы» или «Информационные технологии» создаётся большое количество академических программных продуктов. Оценив уровень качества АПП по выборке за определённое время, преподаватели этих дисциплин получают возможность

сформировать оценку качества всего образовательного процесса и сориентировать усилия на его улучшение. Ввиду этого оценивание качества АПП является важной практической задачей. Для её решения нужен удобный и проверенный инструментарий, базирующийся на надёжных методах.

Задачу разработки методов оценивания качества программных продуктов, специализированных для АПП, нельзя считать полностью решённой в настоящее время. Существует большое разнообразие методов оценивания качества профессиональных программных продуктов (ППП), но дело в том, что АПП существенно отличаются от ППП (см. выше с. 41). Наше исследование [41] учитывает особенности АПП. В нём предлагается методика независимой, то есть, максимально объективной оценки качества АПП, однако инструмент оценивания, разработанный нами ранее, ориентирован лишь на один язык программирования, язык Pascal.

Вопрос распространения этой методики на условия, когда студенты применяют и другие, более современные языки программирования высокого уровня, такие как C, C++, C# или язык MATLAB, решается в настоящее время. С этой целью на основе данной методики создана оригинальная «Программа оценки качественных характеристик академических программных продуктов на основе методики Холстеда» – Halstead method.

### **Методика оценивания качества АПП**

Согласно методике Холстеда [42], оценка качества АПП формируется посредством анализа исходного кода программы и вычисления на этой основе некоторых специальных характеристик кода. Постулируется (см. на с. 42), что текст программы свободен от ошибок, т. е., готов к оцениванию качества.

Результирующие значения параметров качества АПП отыскиваются методами математической статистики в результате обработки большого количества программных текстов, предварительно помещённых в базу данных.

Характеризуем этот процесс детальнее.

Основным параметром качества АПП, согласно данной методике, является критерий совершенства логической структуры текста программного продукта –  $Hq$  [42]. Критерий совершенства  $Hq$  вычисляется на основе четырёх основных характеристик (параметров) программного текста:

- $\eta_1$  – число простых (или отдельных) операторов, появляющихся в данной реализации (словарь операторов);

- $\eta_2$  – число простых (или отдельных) операндов, появляющихся в данной реализации (словарь operandов);
- $N_1$  – общее число всех операторов, появляющихся в данной реализации;
- $N_2$  – общее число всех operandов, появляющихся в данной реализации.

На основе этих параметров для каждого текста АПП вычисляются:

- экспериментально определённая длина

$$N = N_1 + N_2 \quad (1.1)$$

- и теоретическая длина

$$\widehat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2. \quad (1.2)$$

Согласно [41], известно строгое математическое обоснование формулы (1.1), выражающей длину программы  $N$  через число входящих в ее запись операторов  $\eta_1$  и operandов  $\eta_2$ , исходя из минимаксного – для данного типа программ – так называемого «стоимостного» критерия [33]. Доказательство этого основано на предположении, что при создании программы реализуется функциональная задача в максимально экономной и рациональной форме. В работе [33] математически строго выведено модифицированное выражение

$$\widehat{N} = \lambda (\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2) \quad (1.3)$$

и, кроме того, выполнена экспериментальная проверка. В ходе этой проверки обработано десять профессионально составленных программ на языках высокого уровня и в результате получено значение  $\lambda = 0,934336314$ , достаточно близкое к 1. Для коэффициента корреляции  $K$  экспериментально – по выборке из десяти программ – найдено значение  $K = 0,9965818$ .

В литературе по методам «программометрии» используются и другие методы измерения характеристик программ. Применение методик для профессиональных программ приводит к значению  $\lambda$  достаточно близкому к 1, что и обосновывает выбор метрики М. Холстеда в данной работе.

Основная вычислительная нагрузка в нашей методике ложится на статистическую обработку для определения числовых характеристик качества АПП. После того, как все файлы уже обработаны и соответствующие характеристики для каждого из них получены, выполняется вычисление эмпирических статистик для каждого определённого параметра по всей группе АПП. Вычисляются:

1. Математическое ожидание:

$$\widetilde{M}_x = \frac{1}{n} \sum_{i=1}^n x_i. \quad (1.4)$$

2. Дисперсия:

$$\widetilde{D}_x = \frac{1}{n-1} \sum_{i=1}^n (x_i - \widetilde{M}_x)^2. \quad (1.5)$$

3. Среднеквадратичное отклонение:

$$\widetilde{\sigma}_x = \sqrt{\widetilde{D}_x}. \quad (1.6)$$

4. Коэффициент корреляции:

$$\widetilde{r}_{xy} = \frac{1}{(n-1)\widetilde{\sigma}_x\widetilde{\sigma}_y} \sum_{i=1}^n (x_i - \widetilde{M}_x)(y_i - \widetilde{M}_y). \quad (1.7)$$

Коэффициент корреляции (1.7) вычисляется для двух параметров, которыми являются экспериментально определённая длина и теоретически полученная длина АПП.

Критерий совершенства  $Hq$  численно равен коэффициенту корреляции  $\widetilde{r}_{xy}$ :  
 $Hq \triangleq \widetilde{r}_{xy}$ .

Для совершенных программ параметр  $Hq$  близок к 1, для программ, имеющих тот или иной набор несовершенств, отличия от 1 будут более существенны, поэтому в результате коэффициент корреляции  $\widetilde{r}_{xy}$  уменьшится, т. е., уменьшится параметр  $Hq$ .

Параметр  $Hq$  важен для оценивания качества академических программных продуктов, поскольку на начальном этапе создания программ особое внимание уделяется оптимальному построению алгоритма и его правильной реализации на языке программирования.

### Программа вычисления качественных характеристик АПП

Определение качественных характеристик АПП – трудоёмкий и сложный процесс, особенно если учесть, что количество АПП для оценивания велико.

Разработанная программа позволяет преодолеть эти затруднения. Это – специализированная программа для автоматического определения качественных характеристик академических программных продуктов на основе методики, описанной в [41].

Программа реализована в среде Visual Studio на языке программирования C#. Пользователю необходимо выбрать подкаталог файлов – текстов АПП для их дальнейшей обработки, после чего для каждого программного текста происходит формирование и вычисление параметров качества, а именно, таких: словарь операторов  $\eta_1$ , словарь operandов  $\eta_2$ , общее число всех операторов  $N_1$ , общее число всех operandов  $N_2$ , экспериментально полученная длина  $N$  и теоретическая вычисленная длина  $\widehat{N}$ .

Программа считывает посимвольно текст каждого АПП из указанного подкаталога и затем отправляет каждый символ в лексический анализатор. Далее происходит следующее:

- Лексический анализатор на основе поступающих на вход символов формирует лексему, которая затем отправляется в синтаксический анализатор.
- Синтаксический анализатор сопоставляет лексему с грамматикой языка. Он определяет, к какому классу относится входная лексема. Затем он формирует словарь операторов и operandов и вычисляет общее количество операторов и operandов программы.
- Далее программа вычисляет экспериментальную длину и теоретическую длину программного текста.

После того как обработаны все файлы, в программе вычисляется критерий совершенства  $Hq$  для всей группы программных текстов и на экран выводятся результаты работы программы:

- таблица полученных параметров каждого АПП:

$$\eta_1, \eta_2, N_1, N_2, N, \widehat{N};$$

- критерий совершенства  $Hq$ .

### **Состояние программы оценивания качества АПП**

«Программа оценки качественных характеристик академических программных продуктов на основе методики Холстеда» готова к использованию в учебном процессе в Ульяновском государственном университете и также в Ульяновском государственном техническом университете при изучении студентами дисциплин: «Технология программирования», «Языки программирования и методы трансляции», «Методы программирования», «Объектно-ориентированное программирование», «Численные методы», «Методы вычислений» и «Вычислительная математика». Заявка на регистрацию этой программы в декабре 2014 обрабатывается в организации Роспатент.

## 1.4 Положения о выполнении и защите проектов

Данное пособие разделено на четыре части. Часть I, в которой вы сейчас находитесь, – вспомогательная. Следующие три части – основные. В каждой основной части помещены по три проекта. Трудоёмкость проектов сбалансирована между этими частями. Каждая часть может обеспечить лабораторный практикум в течение одного учебного семестра по дисциплине «Вычислительная математика» или по дисциплине «Численные методы». Для этого в учебной программе этих дисциплин должны быть предусмотрены учебные занятия в форме лабораторных работ. Другой приемлемый вариант организации учебного процесса заключается в том, что необходимые часы для выполнения задач лабораторного практикума по дисциплине «Вычислительная математика» или по дисциплине «Численные методы» могут быть предусмотрены в отдельной дисциплине «Практикум на ЭВМ», которая должна идти синхронно с этими дисциплинами (как кореквизит).

Каждый проект, выполняемый в течение семестра, студент защищает, т. е., подаёт пояснительную записку и делает доклад о написанной им компьютерной программе, демонстрируя её работу в дисплейном классе.

Пояснительная записка и устный доклад (защита) оцениваются в баллах отдельно за каждый учебный семестр. Внутри семестра баллы за отдельные защищённые проекты складываются и тем самым образуют общую оценку за этот вид вашей учебной работы, т. е., за лабораторные проекты, разработанные дома и продемонстрированные в классе.

Независимо от того, какую часть лабораторного практикума, представленного в данном пособии, вы реализуете, эта работа занимает один учебный семестр: весенний или осенний.

Рассмотрим **Часть II – Стандартный курс** (с. 85). Здесь установлены следующие сроки сдачи лабораторных проектов:

1. Весенний семестр обучения:

- (а) проект № 1 – с 20 по 31 марта, от 0 до 50 баллов.
- (б) проект № 2 – с 20 по 30 апреля, от 0 до 25 баллов.
- (с) проект № 3 – с 20 по 31 мая, от 0 до 25 баллов.

2. Осенний семестр обучения:

- (а) проект № 1 – с 20 по 31 октября, от 0 до 50 баллов.
- (б) проект № 2 – с 20 по 30 ноября, от 0 до 25 баллов.

(с) проект № 3 – с 20 по 30 декабря, от 0 до 25 баллов.

Если мы рассматриваем **Часть III – Повышенный курс** (с. 315), то график сдачи лабораторных проектов выглядит аналогично:

1. Весенний семестр обучения:

- (а) проект № 4 – с 20 по 31 марта, от 0 до 25 баллов.
- (б) проект № 5 – с 20 по 30 апреля, от 0 до 25 баллов.
- (с) проект № 6 – с 20 по 31 мая, от 0 до 50 баллов.

2. Осенний семестр обучения:

- (а) проект № 4 – с 20 по 31 октября, от 0 до 25 баллов.
- (б) проект № 5 – с 20 по 30 ноября, от 0 до 25 баллов.
- (с) проект № 6 – с 20 по 30 декабря, от 0 до 50 баллов.

Если реализуется **Часть IV – Специальный курс** (с. 377), правила такие же:

1. Весенний семестр обучения:

- (а) проект № 7 – с 20 по 31 марта, от 0 до 25 баллов.
- (б) проект № 8 – с 20 по 30 апреля, от 0 до 25 баллов.
- (с) проект № 9 – с 20 по 31 мая, от 0 до 50 баллов.

2. Осенний семестр обучения:

- (а) проект № 7 – с 20 по 31 октября, от 0 до 25 баллов.
- (б) проект № 8 – с 20 по 30 ноября, от 0 до 25 баллов.
- (с) проект № 9 – с 20 по 30 декабря, от 0 до 50 баллов.

Любая сдача лабораторного задания позже установленного срока влечёт уменьшение вашей общей оценки на 10 баллов. За каждое невыполнено задание начисляется 0 баллов. За все три предлагаемые в семестре лабораторные проекты можно в лучшем случае заработать 100 баллов. Эти максимальные 100 баллов мы распределяем определённым образом между общим числом выданных лабораторных заданий, принимая во внимание их сравнительную трудоёмкость. Если не оговорено другое, предлагаемые студенту проекты являются индивидуальными (негрупповыми).

- Отдельно рассмотрим **Часть I – Стандартный курс**

За проект № 1 можно в лучшем случае заработать 50 баллов (но можно и меньше, если ваш проект или ваша защита проекта обнаружит недостатки); за второй проект – 20 баллов и за третий – максимум 30 баллов. Заработанные баллы существенным образом влияют на экзаменационную, итоговую оценку по всей дисциплине. Более подробно об этом см. здесь в разделе Введение (с. 17) или базовый учебник [6], с. 16–23.

Первые три проекта (с. 85) соответствуют первым трём темам из списка тем:

- тема 1 – методы исключения в решении систем;
- тема 2 – разложения Холесского положительно определённых матриц;
- тема 3 – методы ортогональных преобразований.

Проект № 1 в любом случае должен быть *первым по очереди выполнения*. Это единственное обязательное условие относительно состава выполняемых проектов. Таким образом, проект № 1 – это проект № 1 по выбору преподавателя. Все остальные проекты – это проекты по выбору студента. Это означает, что студент решает самостоятельно, какой проект и сколько проектов отберётся выполнить в течение семестра. Об этом выборе проектов для выполнения студент сообщает преподавателю в начале семестра – в течение первых двух недель, и преподаватель этот выбор студента фиксирует в своём журнале. Во второй половине семестра студент может заявить преподавателю о своём отказе от первоначального выбора, если он увидит, что не справляется, и преподаватель фиксирует «отказ от проекта №...» в своём журнале. Это делается для того, чтобы:

- студент научился правильно оценивать свои способности, планировать своё время и организованно работать для достижения своих целей,
- преподаватель видел, как себя проявляет тот или иной студент, насколько он амбициозен, трудолюбив и способен.

Установлен следующий порядок оценивания проектов в Части I:

За выполненное безупречно и в полном объёме задание по теме № 1 (лабораторный проект № 1) студент заработает 50 баллов. Это задание является базовым и поэтому должно предшествовать всем остальным. Далее, за выполненное безупречно и в полном объёме задание по теме № 2 (лабораторный

проект № 2) студент заработает 20 баллов, а за выполненное безупречно и в полном объёме задание по теме № 3 (лабораторный проект № 3) – 30 баллов.

Для каждого лабораторного проекта необходимо проделать следующее:

1. Написать и отладить программу, реализующую ваш вариант задания.
2. Предусмотреть сообщения, предупреждающие о невозможности решения указанных задач с данной матрицей.
3. Уделить особое внимание эффективности программы (в смысле экономии оперативной памяти).
4. Предусмотреть пошаговое выполнение алгоритмов с выводом результата на экран.
5. Написать пояснительную записку о проделанной работе (см. разд. 6).
6. Представить преподавателю печатный текст пояснительной записи и компакт-диск, содержащий в отдельных директориях весь проект: Программный код проекта (готовый к запуску) + Инструкция пользователя + пояснительная записка в виде \*.doc файла (см. разд. 6).

Защита проекта заключается в демонстрации работоспособной программы в действии в различных режимах проведения вычислительного эксперимента (см. разд. 6) и в докладе о том, как студент разрабатывал программу. Во время доклада преподаватель может попросить студента изменить некоторые фрагменты алгоритма: например, правило вычисления погрешности метода или правило выбора ведущего (главного элемента). Предполагается, что студент сумеет сделать это «на лету»: не только быстро внести изменение в код, но сразу после изменения кода откомпилировать программу, запустить её на исполнение и прокомментировать получаемый при этом результат.

Заработанное число баллов за каждое задание будет уменьшено, если письменный отчёт или устная защита работы не отвечает всем указанным требованиям или не демонстрирует самостоятельное исполнение.



Индивидуальный вариант для каждого студента в проекте № 1 назначается равным  $N + 2$ , где  $N$  – порядковый номер студента в списке группы (в алфавитном порядке).

## 1.5 Заключение по разделу 1

В этом разделе мы изложили особенности проекто-ориентированной методики преподавания и обучения так, как мы видим её возможности применительно к дисциплинам Вычислительной математики и информатики. Сначала характеризован Метод проектов (подразд. 1.1, с. 29) в широком смысле: его зарождение и современное состояние. Показано, что применение этого метода способно ответить на главный вопрос современного высшего образования: КАК изменить к лучшему поведение студента?

Затем изложен Фронтально-состязательный подход (подразд. 1.2, с. 32) – наше представление о том, ЧТО надо сделать для реализации Метода проектов при изучении дисциплин ВМИ, особенно, в условиях больших аудиторий студентов.

В подразделе 1.3 (см. с. 39) мы сообщаем, какой инструментарий нами создаётся для автоматизированного оценивания качества студенческих программных проектов с учётом тех обстоятельств нашей работы в больших потоках студентов, что число оцениваемых проектов велико.

О том, какие правила мы устанавливаем и какие требования предъявляем к самим проектам, студент узнает из прочтения завершающего подраздела 1.3 (см. с. 39).

К сказанному следует добавить, что проекты, представленные в данном учебном пособии, можно рассматривать как «полигон» для освоения студентами не только методов вычислений в линейной алгебре, но гораздо шире – как хороший «полигон» для освоения ими технологии программирования на различных языках высокого уровня, к числу которых принадлежит язык C#. Переходим к изложению элементов этого языка в следующем разделе 2.

## 2

# Элементы программирования на C#

## 2.1 Синтаксис

Рассмотрим базовый синтаксис языка C#, который понадобится вам при выполнении лабораторных проектов, представленных в данном пособии.

### Объявление переменных

Для выполнения лабораторного проекта вам могут потребоваться различные типы переменных. Их объявление показано в примере 2.1.

#### Пример 2.1.

```
int a=0; //целочисленная переменная a
string b=" "; //строковая переменная b
char c=' '; //символьная переменная c
double r=0; //дробная переменная r
bool y=true; //логическая переменная y
//одномерный массив дробных чисел
double[,] mas=new double[<размер>];
//двумерный массив дробных чисел
double[,] mas=new double[<числострока>,<числостолбцов>];
```



Число строк или столбцов может выражаться либо целым числом, либо целочисленной переменной.

### Комментарии

Хорошая программа – это хорошо комментированная программа. Правила включения комментариев просты. Они показаны в примере 2.2.

## Пример 2.2.

```
// Для того чтобы закомментировать строку кода,
// перед ней ставим два слэша.
```



## Некоторые базовые операции

Список основных арифметических операций в C# с примерами использования приведен в табл. 2.1 (см. с. 84).

Кроме приведённых в табл. 2.1 операций, вам могут понадобиться следующие операции: \* (умножение), / (деление), < (меньше) или > (больше). Они работают аналогично. Кроме базовых операций, язык C# предоставляет возможность использования сокращённых операций, состоящих из знака операции и следующего за ним знака присваивания. Вариант использования операции += показан в примере 2.3.

## Пример 2.3.

```
int a=5;
a=a+5; //обычная операция +
a+=5; //аналогичная применённой выше сокращённая операция +=
```



## Цикл while

Это цикл с предусловием (пример 2.4). Он имеет следующий синтаксис:

while (УсловиеПродолжения) {Операторы;}
---

## Пример 2.4.

```
int i=0;
while(i<=10) i++;
```



Цикл работает по следующему правилу: сначала проверяется условие продолжения оператора и в случае, если значение условного выражения равно true, соответствующий оператор (блок операторов) выполняется.

## Цикл do ... while

Цикл с постусловием (пример 2.5). Синтаксис:

do {Операторы;} while(УсловиеПродолжения)

### Пример 2.5.

```
int i=0;  
do  
i++;  
while(i<=10);
```



Отличие от ранее рассмотренного оператора цикла состоит в том, что здесь сначала выполняется оператор (блок операторов), а затем проверяется условие продолжения.

## Цикл for

Пошаговый цикл. Синтаксис заголовка цикла определяется формулой:

for (ВыражениеИнициализации; УсловиеПродолжения; ВыражениеШага)

Здесь

ВыражениеИнициализации,  
УсловиеПродолжения и  
ВыражениеШага  
могут быть пустыми, но наличие пары символов  
;  
в заголовке цикла обязательно.



### Пример 2.6.

```
for(int i=??.;i<??;i++){  
//тело цикла;  
}
```

В примере 2.6 int i=??.; i<?? (или i>??) и i++ (или i--) означают, соответственно, установку начального значения переменной, которая будет изменяться в цикле, установку условия завершения цикла и шаг изменения переменной. Ещё один пример 2.7 показан ниже.

### Пример 2.7.

```
for (int i=0;i<10;i++) Console.WriteLine(i);
```



### Условный оператор if

Условный оператор `if` имеет следующие правила использования. После ключевого слова `if` располагается взятое в круглые скобки условное выражение (булево выражение), следом за которым располагается оператор (блок операторов) произвольной сложности. Далее в операторе `if ... else` после ключевого слова `else` размещается ещё один оператор (блок операторов).

### Пример 2.8.

```
if (i>0)
    Console.WriteLine(i);
else
    Console.WriteLine(i-10);
```



## 2.2 Приложения

### Создание приложений

В Visual Studio на языке C# можно создавать приложения в двух вариантах: консольные приложения и приложения Windows Forms. Первый вариант хорош для тестирования написанных методов, так как не нужно тратить время на проработку интерфейса. Здесь мы обратим особое внимание на методы работы с двумерными массивами.

Второй вариант предназначен для разработки удобных пользовательских приложений. Запуск любых приложений осуществляется командой меню Debug Start→debugging.

Рассмотрим оба варианта подробнее.

### Создание консольного приложения

Для создания консольного приложения в Microsoft Visual Studio необходимо выполнить следующее.

В меню File (Файл) выбрать пункт New (Новый) и в нём выбрать подпункт Project (Проект), как показано на рис. 2.1.

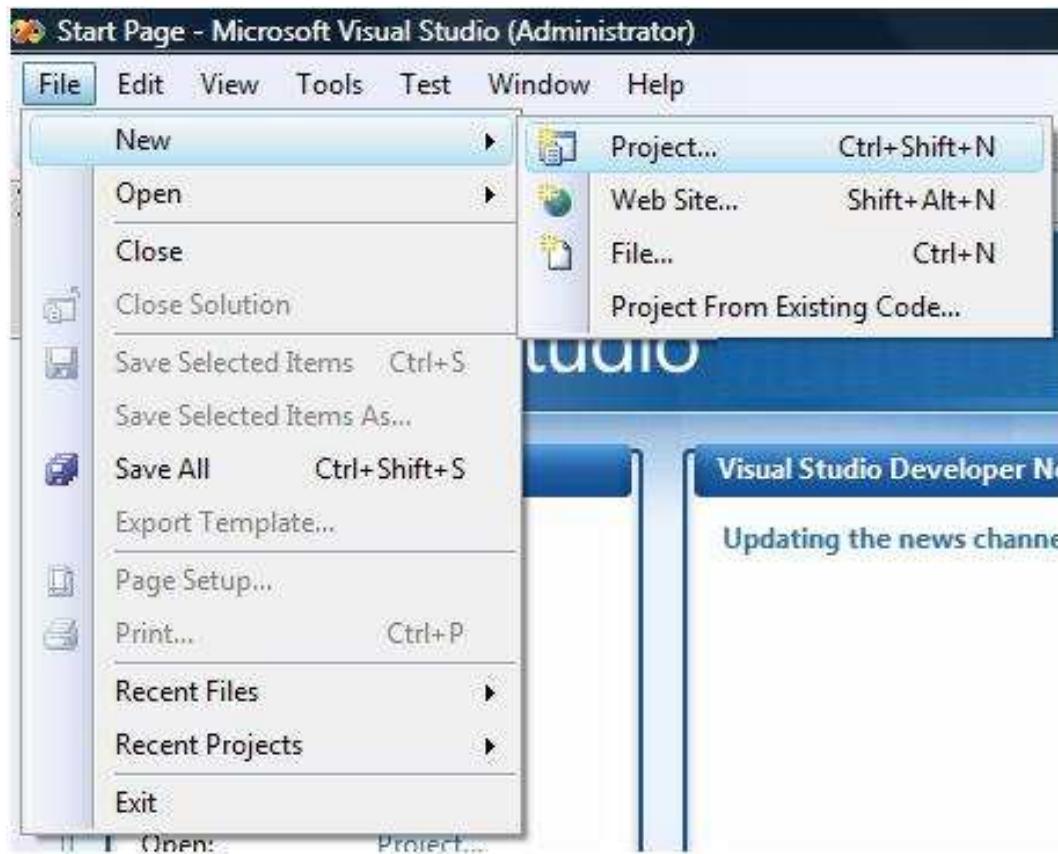


Рис. 2.1. Создание проекта

Далее в диалоговом окне, показанном на рис. 2.2, а, в разделе **Project types** необходимо выбрать пункт **VisualC#** и подпункт **Windows**. В разделе **Templates** выбрать **ConsoleApplication**. В нижней части окна необходимо задать имя проекта и папку, где он будет сохраняться. При этом необходимо следить, чтобы папка, в которую сохраняется проект, была доступна для записи. Если всё сделано правильно, то на экране появится окно, показанное на рис. 2.2, б.

Ввод данных с клавиатуры осуществляется посредством метода **ReadLine()** класса **Console**. Данный метод возвращает строку, которую пользователь ввёл с клавиатуры.

Основное, что необходимо помнить при работе с этим методом, заключается в следующем.

С клавиатуры вы получаете всегда СТРОКУ.

Соответственно, для получения чисел или символов необходимо выполнять преобразования.

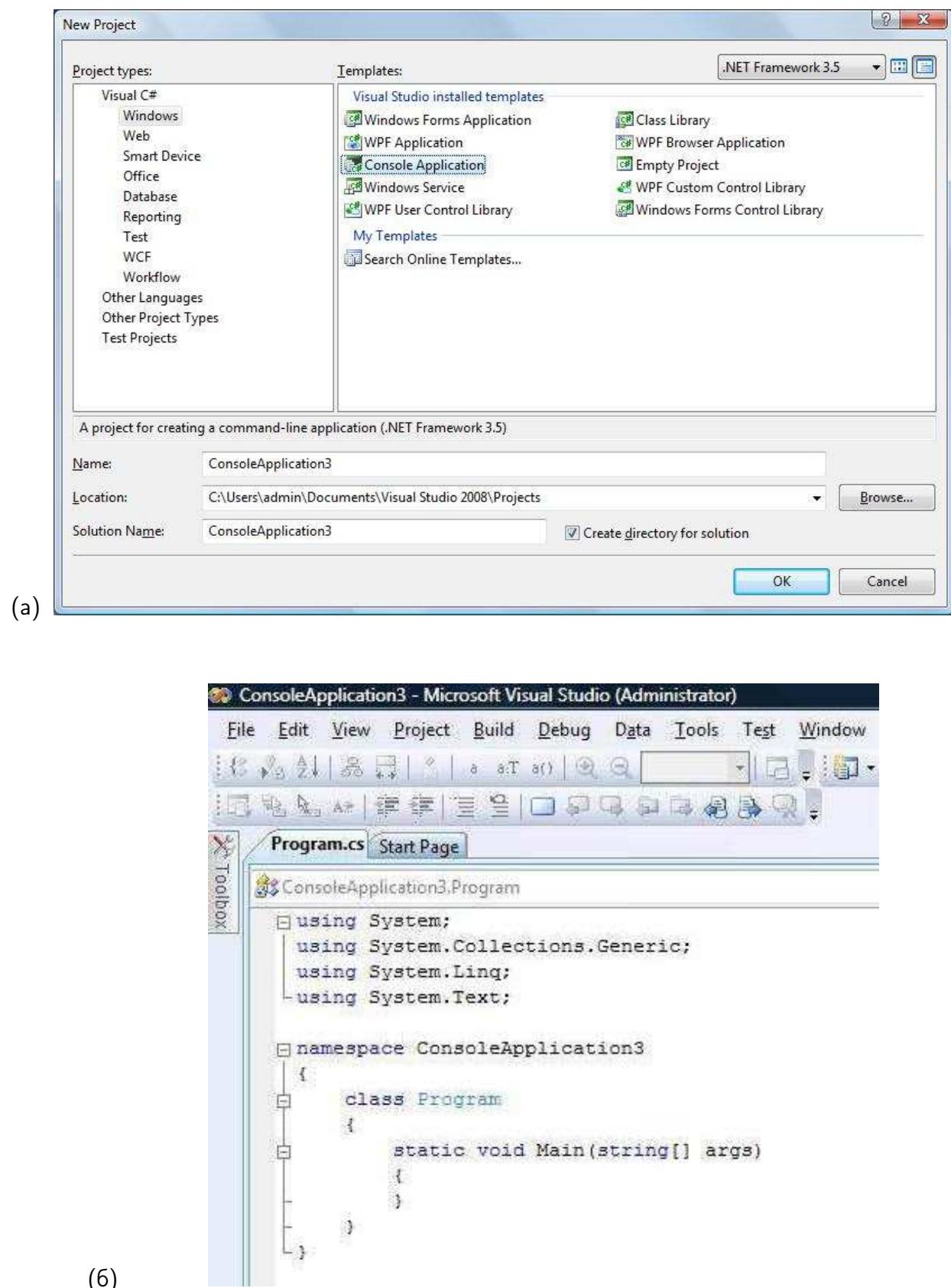


Рис. 2.2. Выбор вида проекта (а). Корректный результат (б)

Получение строки с клавиатуры показано в следующих примерах 2.9 и 2.10.

### Пример 2.9.

```
string s=string.Empty; //сюда мы сохраним введённую информацию  
s=Console.ReadLine(); //непосредственно считывание
```



Можно более кратко:

### Пример 2.10.

```
string s=Console.ReadLine(); //объявление и считывание
```



Получение числа с клавиатуры показано в примерах 2.11 и 2.12

### Пример 2.11.

```
int a=0; //сюда мы сохраним итоговое число  
string s=Console.ReadLine(); //объявление и считывание  
a=Convert.ToInt32(s); //получение числа
```



Можно более кратко:

### Пример 2.12.

```
//объявление, считывание и преобразование  
int a=Convert.ToInt32(Console.ReadLine());
```



С помощью класса Convert можно выполнить преобразования ко всем простым типам данных. Для выполнения преобразования после имени Convert необходимо поставить точку и в появившемся списке найти нужный метод (по имени типа данных), как показано ниже на рис. 2.3.

Вывод данных на экран осуществляется посредством метода `WriteLine()` класса `Console`, то есть осуществляется с помощью конструкции, которая продемонстрирована далее в примере 2.13.

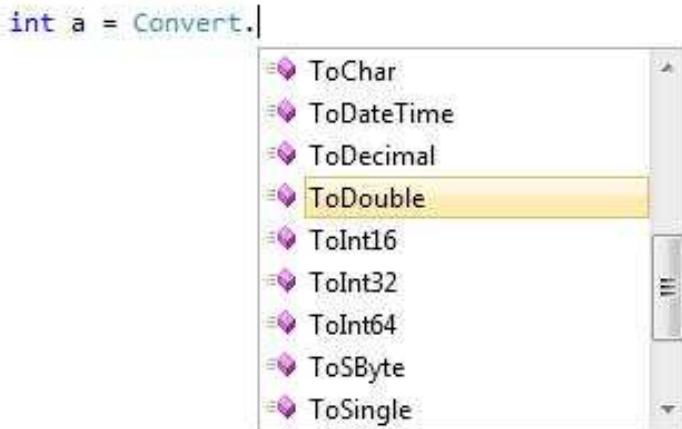


Рис. 2.3. Преобразование типов

**Пример 2.13.**

```
Console.WriteLine(<то, что выводим>);
```

или

```
Console.WriteLine("{0},{1}",<то, что выводим первым>,
                  <то, что выводим вторым>);
```

Здесь <то, что выводим> – либо строковая переменная, либо строка в двойных кавычках. Для вывода числовой переменной следует выполнить преобразование, иллюстрированное в примере 2.14.

**Пример 2.14.**

```
int a=999;
Console.WriteLine("Значение a={0}",a.ToString());
```

Код программы располагаем между фигурными скобками (внутри тела) этого метода по следующему образцу:

```
public static void Main(string[] args) ...
```

Возможный вариант консольного приложения показан в примере 2.15.

### Пример 2.15.

```
using System;
public class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Введите своё имя");
        string st=Console.ReadLine();
        Console.WriteLine("Привет {0}",st);
        Console.ReadLine();
    }
}
```



Для удобства работы в консольном приложении можно организовать текстовое меню. Один из способов показан в примере 2.16.

### Пример 2.16.

```
Console.Clear();
while(true) {
    Console.WriteLine("Ввод данных - 1");
    Console.WriteLine("Обработка данных - 2");
    Console.WriteLine("Вывод данных - 3");
    Console.WriteLine("Выход - 4");
    string param = Console.ReadLine();
    switch(param) {
case "1": input_func(); break;
        case "2": process_func();break;
        case "3": output_func();break;
        case "4": return;
default: Console.WriteLine("Введён некорректный символ.");
        break;
    }
}
```



## Считывание двумерного массива с клавиатуры и вывод его на экран

Так как в лабораторных проектах, предлагаемых в данном учебном пособии, вам придётся работать, в основном, с двумерными массивами, рассмотрим, как считывать двумерный массив с клавиатуры в консольном приложении и как выводить его на экран.

Мы предполагаем, что пользователь вводит корректные элементы массива в строку через пробел, по окончании ввода строки нажимает Enter и переходит к следующей строке. Для окончания ввода массива, вводится знак решетки (#). Соответственно, первое, что нам нужно сделать, – организовать цикл для считывания данных с клавиатуры и определить переменную для хранения считываемой информации, а также определить сам массив. Строковую переменную и пустую ссылку на двумерный массив объявляем так, как показано в примере 2.17.

### Пример 2.17.

```
strings="";
int[,] massiv=null;
```

Затем начинаем цикл с постусловием для считывания данных. В теле этого цикла будем использовать оператор `ReadLine()` и каждую считанную строку будем дописывать в переменную `s`, добавляя в конце восклицательный знак для разделения строк массива. Цикл будет продолжаться, пока пользователь не введёт знак решетки (#), как можно видеть в примере 2.18.

### Пример 2.18.

```
do {
    Console.Write("Введите элементы строки ");
    Console.WriteLine("массива через пробел:");
    stringstr=Console.ReadLine();
    if (str.trim()!="\#")
        s+=str+"!";
} while (str.trim()!="\#");
```

Теперь в строке `s` находятся считанные с клавиатуры данные. Строки массива разделяются знаком «!», а элементы в них – пробелом. На основе этих

данных необходимо сформировать двумерный массив. Выполняем это последовательно в четыре этапа.

1. Объявляем переменную типа массив строк и записываем в неё результат работы метода `Split`, вызванного у переменной `s` (пример 2.19).

### Пример 2.19.

```
string[] stroki=s.Split(newchar[]{','},  
StringSplitOptions.RemoveEmptyEntries);
```



2. Запоминаем в переменную типа `int` размер этого строкового массива; это будет количество строк нашего двумерного массива:

```
int countRow=stroki.Length;
```

3. Организуем цикл перебора элементов строкового массива, применяя к каждому метод `Split` для извлечения чисел. Кроме того, при работе с первым элементом выделяем память под наш двумерный массив. Количество строк мы получили ранее (`countRow`). Количество столбцов будет равно количеству чисел в первом элементе строкового массива. Изначально эта переменная равна нулю; само значение запишем в неё, когда перейдём к первому элементу строкового массива `stroki` (пример 2.20).

### Пример 2.20.

```
int countCol=0;  
for (int i=0;i<countRow;i++){  
    string[] stolbci=stroki[i].Split(newchar[]{','},  
    StringSplitOptions.RemoveEmptyEntries);  
    if (i==0) {  
        countCol=stolbci.Length;  
        massiv=newint [countRow, countCol];  
    }  
}
```



4. Теперь объявим цикл для перебора второго строкового массива и занесения элементов в числовой массив. Так как пользователь мог ввести в разных строках разное количество элементов, нам необходимо будет это учитывать при определении границ цикла перебора. Осуществить это можно с помощью конструкции `Math.Min(a, b)`, где `a` – это переменная `countCol`, а `b` – количество чисел в текущей обрабатываемой строке. Понятно, что в каждой строке числового массива будет столько элементов, сколько пользователь ввёл в первой строке. Таким образом, лишние элементы игнорируются, а недостающие заполняются нулями. В теле второго цикла будем переводить строковый элемент в число и записывать его в числовой массив (пример 2.21).

### Пример 2.21.

```
int countCol=0;
for(int i=0;i<countRow;i++) {
    string[] stolbci=stroki[i].Split(newchar[] {' '},
        StringSplitOptions.RemoveEmptyEntries);
    if (i==0) {
        countCol=stolbci.Length;
        massiv=newint[countRow, countCol];
    }
    for(int j=0;j<Math.Min(countCol,stolbci.Length);j++)
        massiv[i,j]=Convert.ToInt32(stolbci[j]);
}
```

**Замечание 2.1.** Если в массив нужно будет добавлять строки или столбцы, память следует выделять с учётом этих возможных добавлений.

Для генерации массива случайным образом вам понадобится объект класса `Random` и его метод `Next`, возвращающий очередное случайное число в указанном диапазоне чисел.

Для всех лабораторных проектов данного учебного пособия типична следующая ситуация: есть массив заданного размера, и вам необходимо случайным образом заполнить его значениями от 0 до 10. Пример 2.22 показывает возможный вариант решения этой задачи.

### Пример 2.22.

```
Random r=newRandom();
for (int i=0; i<mas.GetLength(0);i++)
    for (int j=0; j<mas.GetLength(1);j++)
        mas[i,j]=r.Next(0,11);
```



Вывод двумерного массива на экран немного осложняется тем, что массив должен быть выведен «табличкой» и элементы должны располагаться строго друг под другом. Для решения этой задачи организуем циклы перебора по строкам и столбцам. В теле второго цикла приводим элемент массива к строке и вызываем у него метод PadLeft с параметром, равным максимальному количеству разрядов вашего числа с учётом знака. Например, если у вас в массиве будут числа в диапазоне от –999 до 999, то максимальное количество разрядов у вас три, а с учётом знака – четыре. Этот метод нужен для того, чтобы числа всегда занимали одно и тоже количество разрядов. Затем выводим этот преобразованный элемент с помощью метода Console.WriteLine, а в теле первого цикла не забываем вызвать Console.WriteLine для перехода на новую строку. Возможный вариант реализации представлен в примере 2.23.

### Пример 2.23.

```
for(int i=0; i<mas.GetLength(0);i++) {
    for(int j=0; j<mas.GetLength(1);j++)
        Console.Write(mas[i,j].ToString().PadLeft(3)+" ");
    Console.WriteLine();
}
```



## Создание приложений WindowsForms

Отличие приложения Windows от консольного приложения заключается в том, что в рамках консольного приложения средством общения с пользователем служит консоль. В приложении Windows с пользователем мы будем общаться посредством визуальных форм.

Для создания приложения Windows в Microsoft Visual Studio необходимо сделать следующее. В меню File (Файл) выбрать пункт New (Новый) и в нём выбрать подпункт Project (Проект). Далее в диалоговом окне, показанном

в разделе *Project types*, необходимо выбрать пункт *Visual C#* и подпункт *Windows*. В разделе *Templates* следует выбрать *Windows Forms Application*.

В результате этих действий на экране появится основная форма, показанная на рис. 2.4, а. На этой форме справа есть вкладка свойств формы *Properties* (если её нет, нажимаем правой кнопкой на форму и выбираем пункт контекстного меню *Properties*). Задаём на ней заголовок формы в поле *Text*, например, «Лабораторная работа (проект) № 1», как показано на рис. 2.4, в.

Для проектирования пользовательского интерфейса используются элементы, размещённые на вкладке *Toolbox*, расположенной справа от формы. Если её там нет, переходим в меню *View Toolbox*. Пользовательский интерфейс можно спроектировать разными способами. Рассмотрим вариант, отличающийся достаточным уровнем удобства для пользователя и относительной простотой реализации для программиста.

Ввод двумерного массива логичнее всего осуществить с помощью таблицы на форме, предварительно запросив у пользователя размер массива. Ввод размера осуществим через два текстовых поля. Для этого переходим слева на вкладку *Toolbox* и выбираем в *CommonControls* элемент *TextBox*, см. рис. 2.4, б.

Размещаем два элемента на форме двойным щелчком мыши или перетаскиванием. Для удобства пользователя разместим несколько элементов *Label* над текстовыми полями и подпишем, что каждое из них будет обозначать: где задаётся количество строк массива, а где – количество столбцов. Шрифты у элементов *Label* настраиваются в *Properties* (свойство *Font*), текст меняется там же (свойство *Text*). Обратите внимание, что вкладка *Properties* отображает свойства текущего (выделенного) элемента. Каждый из этих элементов имеет уникальное имя, по которому к нему можно обратиться из программного кода. Оно отображается жирным шрифтом вверху вкладки *Properties*.

Далее разместим на форме кнопку (*Button*), при нажатии на которую будет отображаться подготовленная для ввода таблица (элемент *DataGridView* из раздела *All Windows Forms*). Теперь форма будет иметь вид, представленный на рис. 2.5.

Теперь сделаем так, чтобы при запуске приложения таблица появлялась не сразу, а лишь после нажатия на кнопку и заполнения соответствующих полей.

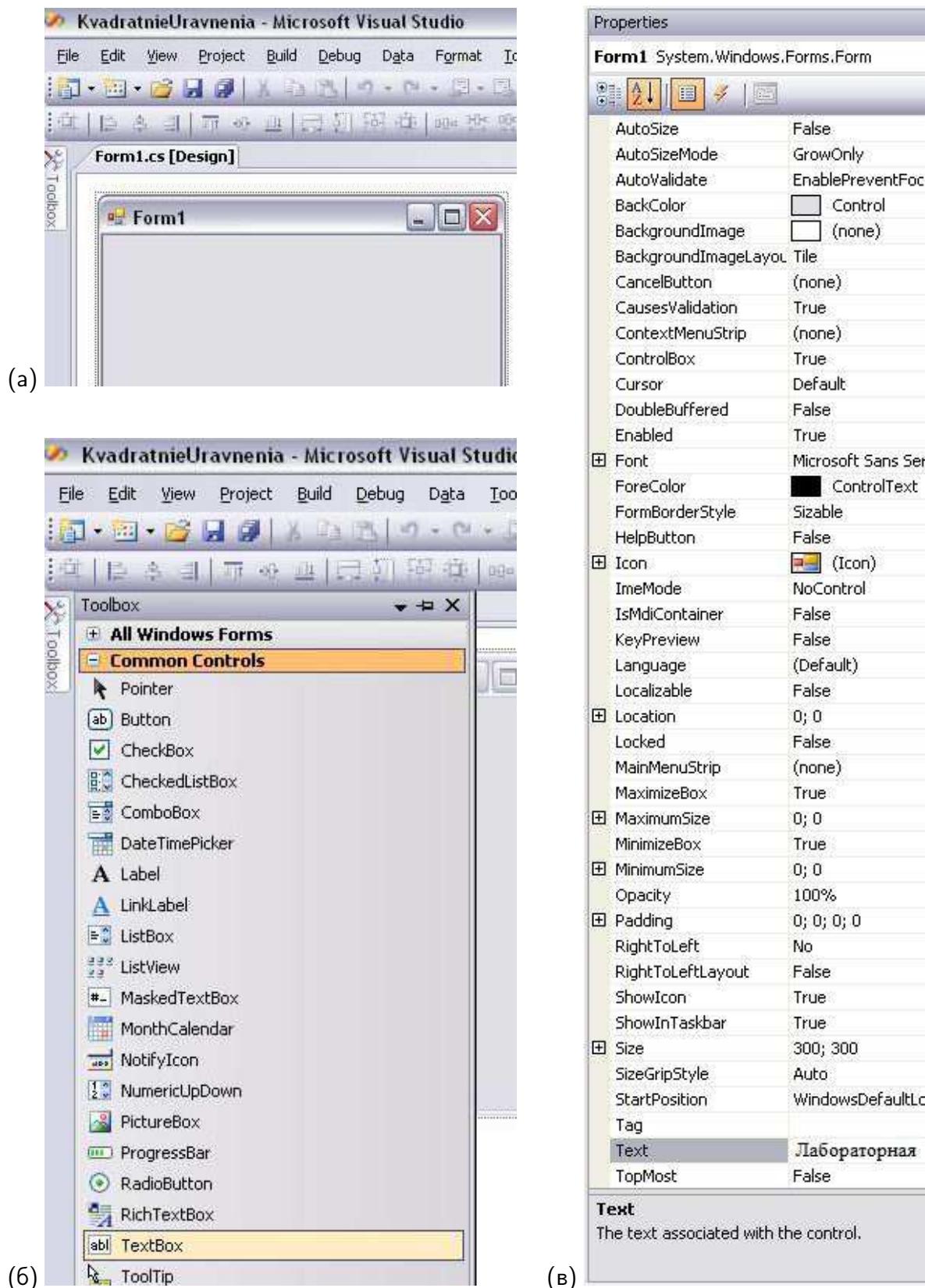


Рис. 2.4. Основная форма (а). Размещение текстовых полей: выбор элемента TextBox в CommonControls (б). Вкладка свойств основной формы (в)

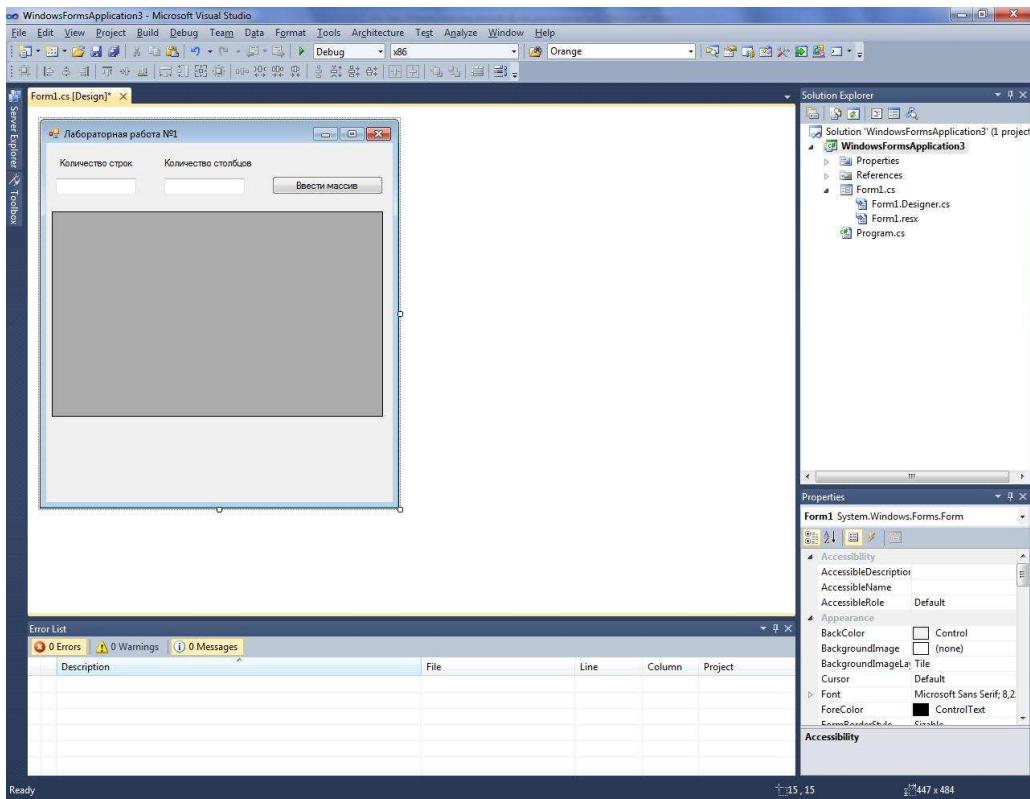


Рис. 2.5. Первый этап создания интерфейса

Для этого переходим во вкладку **Properties** элемента **DataGridView** и устанавливаем для свойства **Visible** значение **false**. Напишем код обработки нажатия кнопки, меняющей свойство видимости таблицы. Этот код размещается в файле **Form1.cs** после двойного нажатия мышкой на кнопку (рис. 2.6, а).

Теперь нам нужно создать массив, расчертить таблицу и по нажатию второй кнопки считать информацию из таблицы в объявленный массив. Начнём с таблицы.

Для начала ограничим для пользователя действия, касающиеся изменения структуры таблицы. Для этого щёлкнем на чёрный треугольник в правом верхнем углу элемента и уберём все галочки, кроме **EnableEditing**, как показано на рис. 2.6, б.

Затем зададим количество строк и количество столбцов таблицы равными введённым в текстовые поля числам, прописав соответствующий код в обработке нажатия на первую кнопку, перед изменением свойства видимости таблицы. Также настроим ширину столбцов, чтобы таблица не занимала слишком много места.

В результате должен получиться метод, показанный в примере 2.24.

## 2 Элементы программирования C#

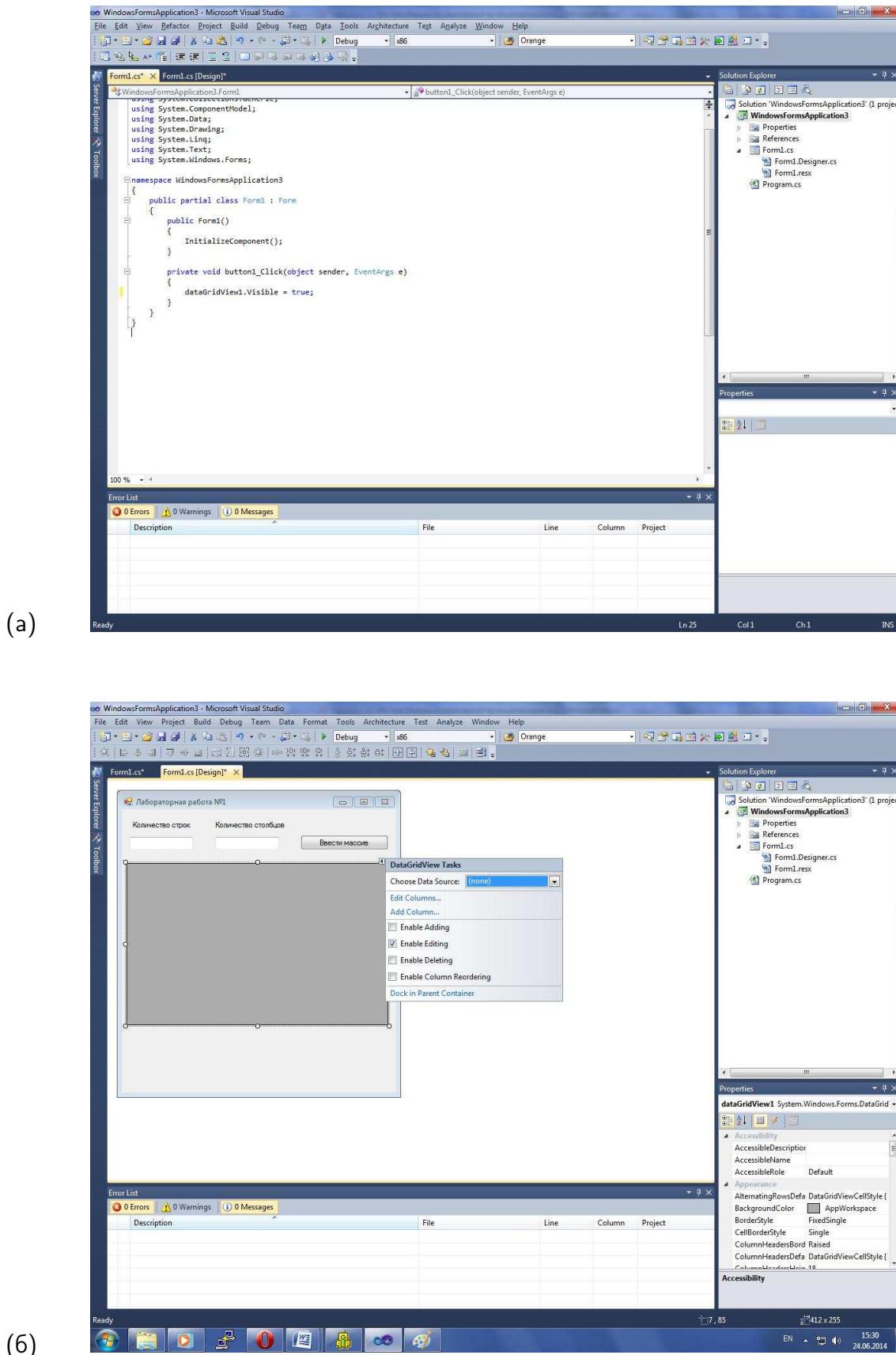


Рис. 2.6. Обработка нажатия на кнопку (а). Настройка параметров таблицы (б)

**Пример 2.24.**

```
private void button1_Click(object sender, EventArgs e) {
    dataGridView1.ColumnCount=Convert.ToInt32(textBox2.Text);
    dataGridView1.RowCount=Convert.ToInt32(textBox1.Text);
    for(int i=0; i<dataGridView1.ColumnCount;i++)
        dataGridView1.Columns[i].Width=50;
    dataGridView1.Visible=true;
}
```



Теперь объявим массив перед созданием метода обработки нажатия на кнопку. Затем разместим на форме ещё одну кнопку и для обработки нажатия на неё напишем код считывания данных из таблицы в объявленный массив.

Разместим на форме ещё один элемент dataGridView и при нажатии той же кнопки выведем в него считанный массив для проверки. Затем разместим на форме textBox для вывода справочной информации. Сделаем его многострочным, установив значение свойства MultiLine в true и растянув его в высоту. Для примера здесь же запишем в него количество строк и столбцов массива.

В итоге получим код, показанный в примере 2.25.

**Пример 2.25.**

```
public partial class Form1: Form
{
    public Form1() { InitializeComponent(); }
    double[,] mas=null;
    private void button1_Click(object sender, EventArgs e) {
        dataGridView1.ColumnCount=Convert.ToInt32(textBox2.Text);
        dataGridView1.RowCount=Convert.ToInt32(textBox1.Text);
        for(int i=0;i<dataGridView1.ColumnCount;i++)
            dataGridView1.Columns[i].Width=50;
        dataGridView1.Visible=true;
    }
    private void button2_Click(object sender, EventArgs e) {
        mas=new double[dataGridView1.RowCount,
                        dataGridView1.ColumnCount];
        for(int i=0; i<dataGridView1.RowCount; i++)
            for(int j=0; j<dataGridView1.ColumnCount; j++)
                {string s= dataGridView1.Rows[i].Cells[j].Value.ToString();

```

```
        mas[i,j]=Convert.ToDouble(s);}  
        dataGridView2.ColumnCount = mas.GetLength(0);  
dataGridView2.RowCount = mas.GetLength(1);  
for(int i=0; i<dataGridView2.RowCount; i++)  
    for(int j=0; j<dataGridView2.ColumnCount; j++) {  
        dataGridView2.Columns[j].Width=50;  
        dataGridView2.Rows[i].Cells[j].Value=mas[i,j];  
    }  
dataGridView2.Visible=true;  
textBox3.Text="Строк в массиве: "+  
            mas.GetLength(0)+Environment.NewLine;  
textBox3.Text+="Столбцов в массиве:  
"+mas.GetLength(1)+Environment.NewLine;  
}  
}
```



Обратите внимание на следующее:

- Команда `Environment.NewLine` в приведённом коде осуществляет перевод каретки на следующую строку.
- В качестве разделителя дробной части используется запятая.

В результате проделанной работы ввод массива из двух строк и двух столбцов будет выглядеть, как показано на рис. 2.7.

Таким образом, мы рассмотрели основные методы работы с данными, необходимые для выполнения лабораторных проектов.

## 2.3 Сервис

### Построение графиков средствами C#

Графики средствами C# можно построить разными способами: с помощью встроенных средств или с использованием сторонних библиотек. Мы рассмотрим построение графиков возможностями технологии GDI+.

GDI (Graphics Device Interface) является интерфейсом Windows, предназначенным для представления графических объектов и вывод их на монитор



Рис. 2.7. Работа приложения

или принтер. На базе технологии GDI была разработана GDI+. Это улучшенная среда для 2D-графики, расширенная возможностями сглаживания линий, использования координат с плавающей точкой, градиентной заливки, использованием ARGB-цветов и т. п.

В данном пособии мы рассмотрим технику рисования в оперативной памяти с последующим выводом на форму. Для рисования мы будем использовать объект класса `Bitmap`, а для вывода на форму – элемент управления `PictureBox`. Рассмотрим написание простейшего примера, когда при нажатии на кнопку на форме будет отображаться график некоторой функции. Размещаем на форме `PictureBox` и `Button`. Переходим к методу обработки нажатия на кнопку. Сначала нам нужно создать объект класса `Bitmap`, с размерами, совпадающими с `PictureBox`. В данном случае `Bitmap` выполняет роль холста, а `PictureBox` – рамки, в которую этот холст повесят. Поэтому размеры должны совпадать. Результатом является следующий код:

```
Bitmap bmp=new Bitmap(pictureBox1.Width, pictureBox1.Height);
```

Далее мы должны создать объект класса `Graphics` – основного класса, предоставляющего доступ к возможностям GDI+. Для данного класса не определено ни одного конструктора. Его объект создаётся в ходе выполнения ряда

методов применительно к конкретным объектам, у которых есть поверхность для рисования. Одним из таких объектов и является Bitmap. Поэтому создаём объект класса Graphics следующим образом:

```
Graphics gr=Graphics.FromImage(bmp);
```

Теперь все вызовы методов отображения фигур будут отрабатывать на нашей битовой карте. Класс Graphics содержит множество методов рисования вида Fill\* или Draw\*, отвечающих за отображение закрашенных или незакрашенных фигур. Первая группа методов в качестве одного из параметров принимает объект типа Brush (кисть), а вторая – объект типа Pen (карандаш). Исключение – метод DrawString, который отображает текст. Этот метод в качестве одного из параметров принимает объект Brush. В том же обработчике объявим массив типа PointF, в котором будут храниться координаты 25 точек нашей функции:

```
PointF[] mas=new PointF[25];
```

Заполним его случайными значениями по у и от 0 до 24 по x, согласно примеру 2.26.

### Пример 2.26.

```
Random r=new Random();
for(int i=0;i< mas.Length;i++)
    mas[i]=new PointF(i,(float)Math.Round(r.Next(0,10)+r.NextDouble(),2));
```



Чтобы график красиво отображался на форме, подстраиваясь под её размер, нужно ввести коэффициенты перевода реальных координат в экранные. Найдём максимальные значения x и y одним из известных способов. Сохраним эти координаты в переменных maxX и maxY, соответственно. Введём два коэффициента перевода координат, рассчитываемых по формулам примера 2.27.

### Пример 2.27.

```
float cdx = (w - 90) / maxX;
float cdy = (h - 50) / maxY;
```



Здесь  $w$  – ширина `pictureBox`, а  $h$  – высота; 90 и 50 – отступы, необходимые для того, чтобы оси координат и подписи не упирались в края. Так как в GDI+ ось  $y$  начинается из левого верхнего угла, нам необходимо будет инвертировать координаты. Для этого введём дополнительную переменную

```
int max = h-25
```

Чтобы нарисовать оси координат, создадим переменную-карандаш, изображающую линии со стрелками на концах, как показано в примере 2.28.

### Пример 2.28.

```
Pen p = new Pen(Brushes.Green, 3);
p.EndCap = System.Drawing.Drawing2D.LineCap.ArrowAnchor;
```



Используя эту переменную, изобразим оси  $x$  и  $y$ :

```
gr.DrawLine(p, (int)(0 * cdx)+25, (int)h-25, (int)(0 * cdx)+25, 0);
gr.DrawLine(p, 0, (int)(max-0*cdy), (int)w-25, (int)(max-0*cdy));
```

Видно, что здесь используется метод `DrawLine`. В качестве параметров он принимает переменную-карандаш, а затем координаты  $x$ ,  $y$  начальной и  $x$ ,  $y$  конечной точек, которые нужно соединить линией. Координаты должны иметь тип `int`, поэтому используем явное приведение к этому типу. Хотя в этом случае приведение можно опустить, но для того, чтобы в дальнейшем вывод осуществлялся по этому шаблону, мы намеренно его усложнили.

Далее расчертим координатную сетку и выведем надписи: по оси  $x$ , как показано в примере 2.29, и по оси  $y$ , как показано в примере 2.30.

### Пример 2.29.

```
for (float x = 0; x <= maxX+1; x++){
    gr.DrawString(x.ToString(), new Font("Arial", 10),
                  Brushes.Green, (int)(x*cdx)+25, max+10);
    gr.DrawLine(Pens.Green, (int)(x*cdx)+25, (int) h-25,
               (int)(x*cdx)+25, 0);
}
```



### Пример 2.30.

```
for (float y = 1; y <= maxY + 1; y++) {  
    gr.DrawString(y.ToString(), new Font("Arial", 10),  
        Brushes.Green, 0, (int)(max-y*cdy));  
    gr.DrawLine(Pens.Green, 0, (int)(max-(int)(y*cdy)),  
        (int) w-25, (int)(max-(int)(y*cdy)));  
}
```



Теперь займёмся отображением графика. Каждую точку мы будем отмечать красным кружком, а затем соединять линией. Поэтому сначала рисуем красный кружок, соответствующий первой точке графика, как показано в примере 2.31.

### Пример 2.31.

```
gr.FillEllipse(Brushes.Red, (int)(mas[0].X*cdx)+25,  
    (int)(max-mas[0].Y*cdy), 5, 5);
```



Затем организуем цикл прохода по массиву и, кроме вывода красной точки, будем соединять предыдущие точки с текущими, как показано в следующем примере 2.32.

### Пример 2.32.

```
for (int i = 1; i < mas.Length; i++) {  
    gr.FillEllipse(Brushes.Red, (int)(mas[i].X*cdx)+25,  
        (int)(max-mas[i].Y*cdy), 5, 5);  
    gr.DrawLine(new Pen(Brushes.Black, 2), (int)(mas[i].X*cdx)+25,  
        (int)(max-mas[i].Y*cdy), (int)(mas[i-1].X*cdx)+25,  
        (int)(max-mas[i-1].Y*cdy));  
}
```



Если сейчас запустить приложение и нажать на кнопку, то ничего не произойдёт. Вернее, график будет нарисован, но мы его не увидим, так как сейчас он находится в оперативной памяти. Для того чтобы отобразить нарисованную картинку на форме, необходимо добавить следующую строку кода:

```
pictureBox1.Image=bmp;
```

В результате на экране отобразится график, вид которого показан на рис. 2.8.

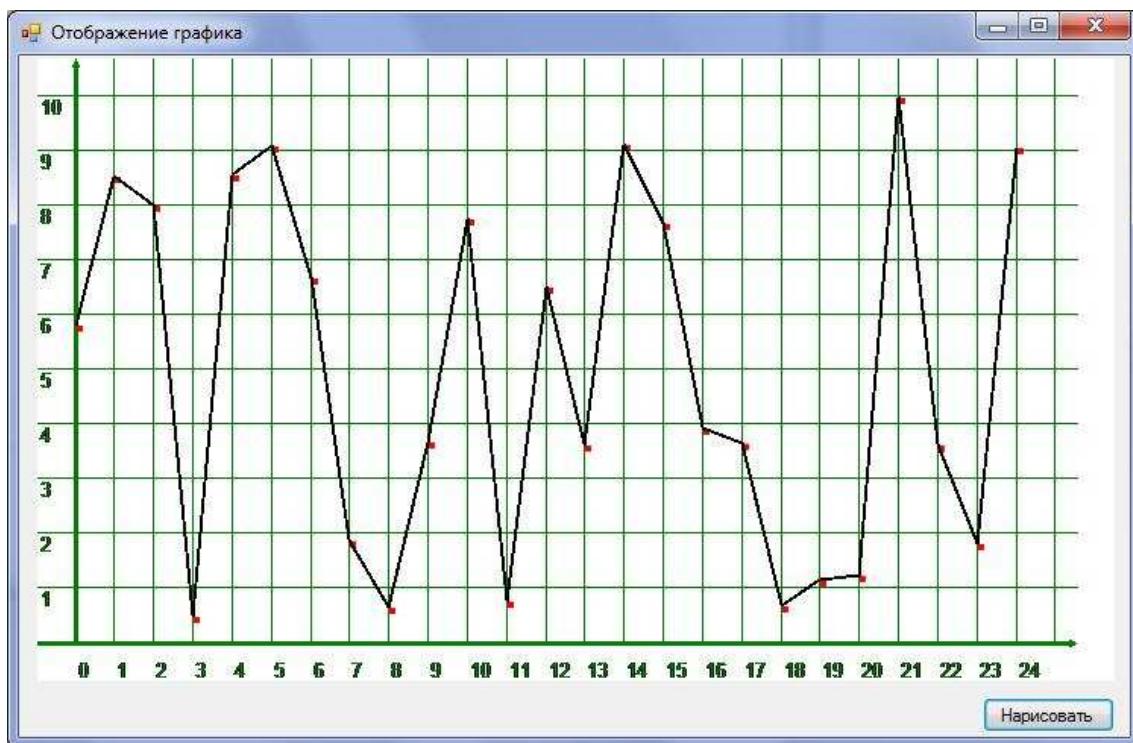


Рис. 2.8. Отображение графика

Если график необходимо вывести в виде гистограммы, то вместо рисования красных точек и линий нужно выводить прямоугольники:

### Пример 2.33.

```
for (int i=0; i<mas.Length; i++)
    gr.FillRectangle(Brushes.Red, (int)(mas[i].X*cdx)+25,
                     (int)(max-mas[i].Y*cdy), 10, (int)(mas[i].Y*cdy));
```

Код примера 2.33 позволяет получить график в виде гистограммы (рис. 2.9).

Обратите внимание на то, что данный алгоритм годится только для построения графиков с положительными координатами точек. Для вывода графиков с отрицательными координатами потребуются дополнительные преобразования.

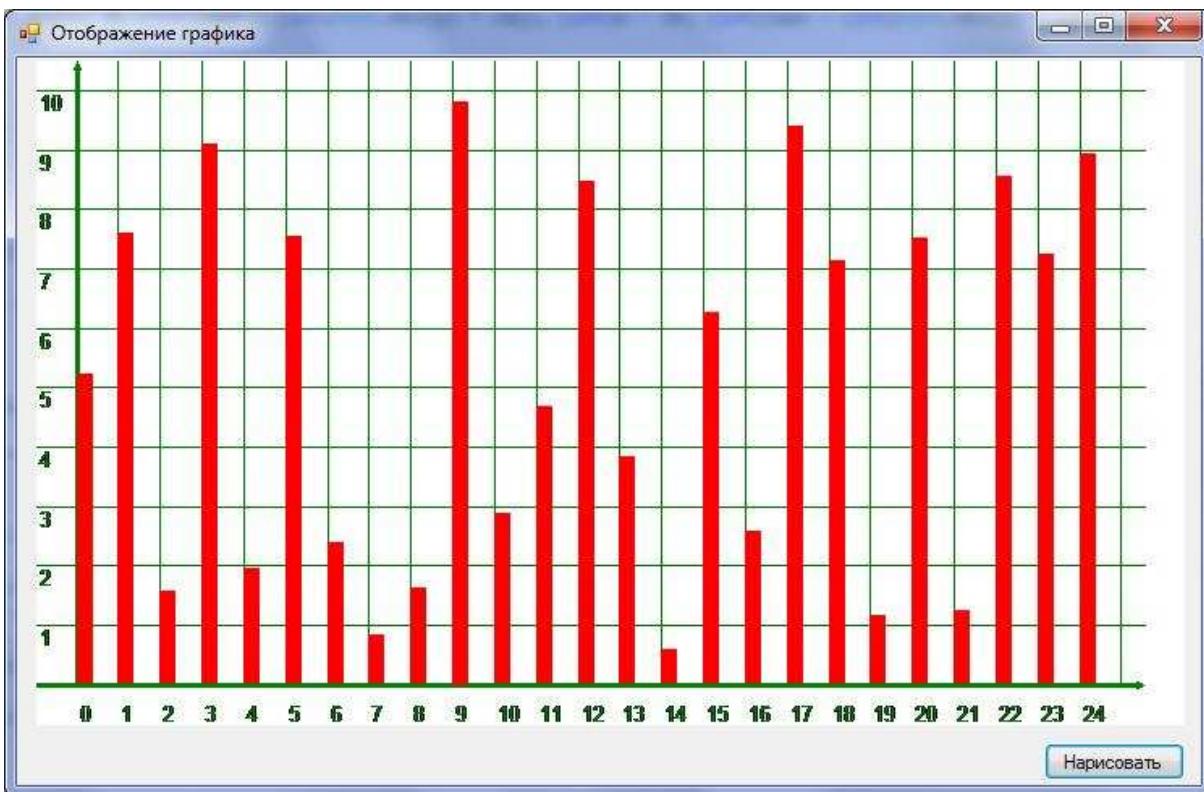


Рис. 2.9. Отображение гистограммы

## Сохранение и загрузка результатов эксперимента

Расширим нашу программу, добавив в неё возможности сохранения и загрузки результатов экспериментов. Для этого, конечно, можно использовать простую запись в текстовый файл. Однако этот метод весьма неудобный, и он требует пристального внимания: запись и чтение информации должны выполняться в строго определённой последовательности с использованием либо ключевых символов, либо чёткого разделения по строкам. Гораздо проще использовать механизм XML-сериализации, предоставляемый C#. В данном случае под сериализацией будем понимать процесс перевода какого-либо объекта в двоичный файл на диске, например, в XML-файл. Восстановление объекта из файла называется десериализацией.

По существу, XML представляет собой текстовый формат, предназначенный для хранения структурированных данных. Выгода от его использования обуславливается следующим:

- в этом формате легко могут быть описаны такие структуры данных, как записи, списки и деревья;

- этот формат представляет собой простой текст, свободный от лицензирования и каких-либо ограничений; кроме того, он не зависит от платформы;
- в C# для работы с этим форматом существуют очень удобные инструменты.

Рассмотрим пример программного кода, позволяющий получить подобный файл. Для реализации используем возможности класса `XMLSerializer`. Выполнение функций сериализации и десериализации этот класс осуществляет методами `Serialize` и `Deserialize`, соответственно. Первый метод является процедурным, принимающим в качестве параметров указатель на файловый поток для записи, и также объект, который необходимо сериализовать. Второй метод является функциональным. Он принимает указатель на файловый поток, откуда осуществляется чтение ранее сериализованного объекта; в результате своей работы он возвращает объект, если чтение прошло корректно, либо пустую ссылку – в противном случае.

Первое, что необходимо помнить, это то, что такой способ работает только с открытыми типами и открытыми членами этих типов. Второе, – мы сериализуем и десериализуем всегда только объект. Поэтому сначала необходимо выделить классы-сущности, которые будут использоваться в приложении. По существу, их два: тест и тестируемый, но так как необходимо хранить ещё и сам список тестируемых, понадобится дополнительный класс. Третье, что необходимо помнить, – сериализуемый объект должен обязательно иметь конструктор без параметров, даже если в программном коде он не будет использоваться.

Допустим, для каждого эксперимента нам нужно сохранить его название, время проведения и массив точек для графика. Тогда мы описываем класс, характеризующий этот эксперимент. Для этого переходим в меню `Project` (Проект) и выбираем пункт `Add class` (Добавить класс). В итоге должно появиться окно (рис. 2.10).

Записываем имя класса `Experiment.cs` и нажимаем кнопку `Add` (Добавить). В открывшемся файле пишем следующий код:

### Пример 2.34.

```
public class Experiment
{
    public float durationTime=0;
    public string name="";
    public PointF[] mas;
}
```

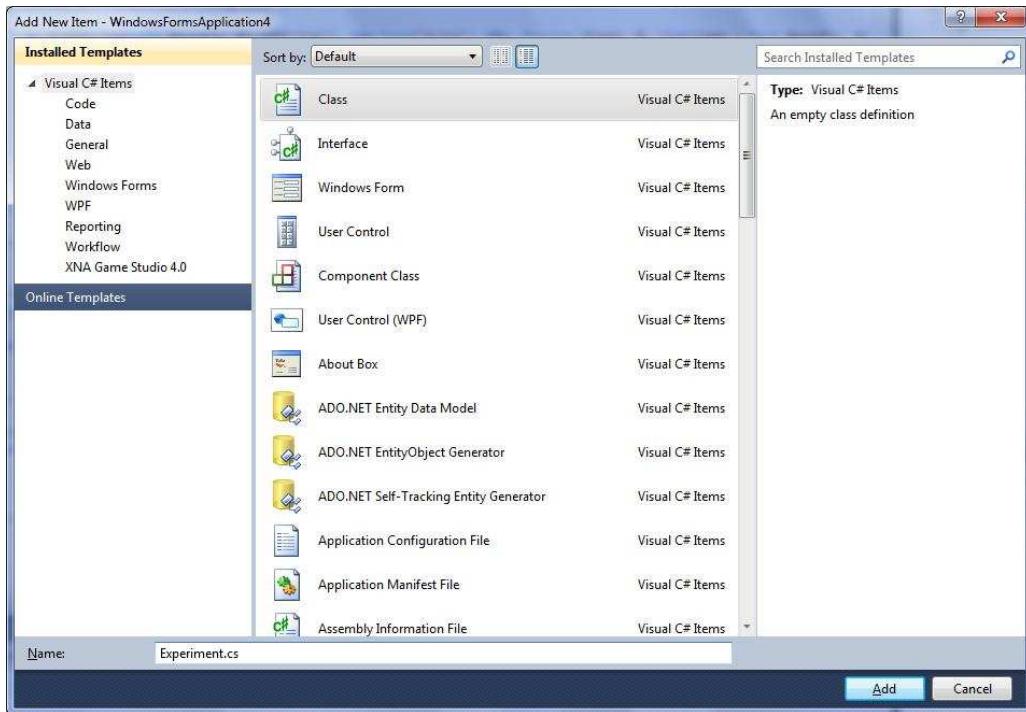


Рис. 2.10. Добавление класса

Для того чтобы тип `PointF` стал доступен в этом классе, в первой строке файла добавляем строку подключения необходимого пространства имён:

```
using System.Drawing;
```

Теперь на форме создаем поле – объект класса `Experiment`. Код объявления пишем перед методом обработки нажатия на первую кнопку. В самом же методе меняем обращение к массиву, то есть, теперь используем не локальную переменную, а поле нашего объекта-эксперимента. В итоге должен получиться следующий код:

### Пример 2.35.

```
Experiment ex=new Experiment();
private void button1_Click(object sender, EventArgs e) {
    Bitmap bmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
    Graphics gr = Graphics.FromImage(bmp);
    ex.mas=new PointF[25];

    Random r=new Random();
    for (int i=0;i< ex.mas.Length;i++)
        ex.mas[i]=new PointF(i, (float)Math.Round(r.Next(0, 10)+r.NextDouble(), 2));
```

```
//код объявления и нахождения maxX и maxY
    int w= pictureBox1.Width;
    int h=pictureBox1.Height;

    float cdx = (w - 90) / maxX;
    float cdy = (h - 50) / maxY;

    int max = h-25;

    Pen p = new Pen(Brushes.Green, 3);
    p.EndCap = System.Drawing.Drawing2D.LineCap.ArrowAnchor;

    gr.DrawLine(p, (int)(0*cdx)+25, (int)h-25, (int)(0*cdx)+25, 0);
    gr.DrawLine(p, 0, (int)(max-0*cdy), (int)w-25, (int)(max-0*cdy));

    for (float x=0; x<=maxX+1; x++) {
        gr.DrawString(x.ToString(),newFont("Arial", 10),
                      Brushes.Green, (int)(x*cdx)+25, max+10);
        gr.DrawLine(Pens.Green, (int)(x*cdx)+25,
                   (int)h-25, (int)(x*cdx)+25, 0);
    }

    for (float y=1; y<=maxY+1; y++) {
        gr.DrawString(y.ToString(),newFont("Arial", 10),
                      Brushes.Green, 0, (int)(max-y*cdy));
        gr.DrawLine(Pens.Green, 0, (int)(max-(int)(y*cdy)),
                   (int) w-25, (int)(max-(int)(y*cdy)));
    }

    gr.FillEllipse(Brushes.Red, (int)(ex.mas[0].X*cdx)+25,
                  (int)(max-ex.mas[0].Y*cdy), 5, 5);

    for (int i=1; i<ex.mas.Length; i++) {
        gr.FillEllipse(Brushes.Red, (int)(ex.mas[i].X*cdx)+25,
                      (int)(max-ex.mas[i].Y*cdy), 5, 5);
        gr.DrawLine(new Pen(Brushes.Black,2),
                   (int)(ex.mas[i].X*cdx)+25, (int)(max-ex.mas[i].Y*cdy),
                   (int)(ex.mas[i-1].X*cdx)+25, (int)(max-ex.mas[i-1].Y*cdy));
    }
```

```
        }
        pictureBox1.Image = bmp;
    }
```



Разместим на форме два элемента textBox для введения имени эксперимента и его длительности. Разместим на форме кнопку и назовем её «Создание эксперимента». В методе её обработки напишем код заполнения полей эксперимента, не забывая переместить (именно переместить, а не скопировать!) из первой кнопки код заполнения массива. В результате должно получиться следующее:

### Пример 2.36.

```
private void button2_Click(object sender, EventArgs e) {
    ex.name(textBox1.Text);
    ex.durationTime=float.Parse(textBox2.Text);
    ex.mas= new PointF[25];
    Random r= new Random();
    for (int i=0;i<ex.mas.Length;i++)
        ex.mas[i]= new PointF(i, (float)Math.Round(r.Next(0, 10) +
                                         r.NextDouble(), 2));
}
```



Чтобы отобразить график, необходимо сначала заполнить текстовые поля и нажать на эту кнопку для генерации данных эксперимента. В противном случае программа выдаст сообщение об ошибке.

Перейдём к сериализации и десериализации. В первой строке файла добавим ссылки на необходимые пространства имён:

### Пример 2.37.

```
using System.Xml;
using System.Xml.Serialization;
using System.IO;
```



Разместим на форме ещё две кнопки: для сохранения эксперимента в файл и его загрузки из файла. Напишем код обработчика кнопки «Сохранение».

Для этого сначала создадим объект класса `XmlSerializer` и настроим его на нужный нам тип данных:

```
XmlSerializer serializer = new XmlSerializer(typeof(Experiment));
```

Теперь создадим объект типа `SaveFileDialog`, с помощью которого будем выбирать файл, куда нужно сохранить информацию:

```
SaveFileDialog sv = new SaveFileDialog();
```

Затем, обезопасив себя от исключений, запустим диалог и на случай, если пользователь выбрал файл, создадим файловый поток и запишем в него информацию об объекте. Если всё прошло успешно, выведем сообщение «Выполнено!», иначе выведем сообщение о неудачной сериализации:

### Пример 2.38.

```
try{
    if (sv.ShowDialog() == System.Windows.Forms.DialogResult.OK) {
        FileStream f = new FileStream(sv.FileName, FileMode.OpenOrCreate);
        using (StreamWriter sw = new StreamWriter(f)) {
            serializer.Serialize(sw, ex);
        }
        MessageBox.Show("Выполнено!");
    }
    else MessageBox.Show("Сериализации не произошло!");
}

catch (Exception excp) {
    MessageBox.Show("Сериализации не произошло! " + excp.Message);
}
```



Если всё сделано верно, то после заполнения текстовых полей, нажатия на вторую, а затем и на третью кнопку должен открыться стандартный диалог выбора файла для сохранения, а после выбора файла на диске должен появиться файл, примерное содержание которого показано в примере 2.39:

### Пример 2.39.

```
<?xmlversion="1.0" encoding="utf-8"?>
<Experiment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<durationTime>100</durationTime>
<name>Эксперимент 1</name>
<mas>
    <PointF>
        <X>0</X>
        <Y>8.36</Y>
    </PointF>
    ...
    <PointF>
        <X>1</X>
        <Y>9.68</Y>
    </PointF>
</mas>
</Experiment>
```



Теперь напишем код загрузки эксперимента из файла. Для этого в кнопке загрузки создадим объект-сериализатор, но вместо диалога сохранения выведем диалог открытия. В случае неудачной загрузки дополнительно обезопасим себя проверкой на пустую ссылку и выводом соответствующего сообщения. В итоге обработчик кнопки будет содержать следующий код:

#### Пример 2.40.

```
XmlSerializer serializer = new XmlSerializer(typeof(Experiment));
OpenFileDialog sv = new OpenFileDialog();
try{
    if((sv.ShowDialog() == System.Windows.Forms.DialogResult.OK) &&
       (File.Exists(sv.FileName))) {
        FileStream f = new FileStream(sv.FileName, FileMode.Open);
        using(StreamReader sw = new StreamReader(f)) {
            ex= serializer.Deserialize(sw) as Experiment;
        }
        MessageBox.Show("Выполнено!");
    }
    else MessageBox.Show("Десериализации не произошло!");
}
catch (Exception excp) {
    MessageBox.Show("Десериализации не произошло! "+excp.Message);
```

```
}

if(ex == null) {
    ex = new Experiment();
    MessageBox.Show("Десериализации не произошло!");
}
```



## Вычисление времени эксперимента

Создадим в тексте описания формы поле типа `DateTime` и в начале кода эксперимента запишем туда текущее время:

```
DateTime dt = DateTime.Now;
```

После окончания эксперимента выведем сообщение о времени эксперимента:

```
MessageBox.Show((DateTime.Now - dt).ToString());
```

Обратите внимание на то, что время выводится в формате

«ЧЧ:ММ:СС.милисекунды».

## 2.4 Заключение по разделу 2

В этом разделе мы рассмотрели основы синтаксиса, создание приложений и сервисных программ на языке программирования высокого уровня C#.

Эти вопросы мы проиллюстрировали множеством полезных примеров, которые помогут студентам выполнить собственные лабораторные проекты. Применённые приёмы не являются ни обязательными, ни единственными верными. Здесь представлен только один из вариантов решения. Каждый студент найдёт самостоятельное решение всех практических вопросов, которые будут у него возникать по ходу разработки проекта.

Наиболее полное представление об объёме работы, требуемой при этом для создания качественного программного продукта, можно получить после ознакомления с полной версией одного из реальных проектов в разд. 6.

## Приложение к разд. 2

Таблица 2.1. Арифметические операции – иллюстрация для текста на с. 53

Знак операции	Назначение	Пример использования
+	Сложение	Запишем в переменную с результат сложения значений а и у: <pre>int a=10; int y=12; int c=a+y;</pre>
-	Вычитание	Запишем в переменную с результат вычитания у из а: <pre>int a=10; int y=12; int c=a-y;</pre>
++	Увеличение на 1	Увеличим переменную с на 1: <pre>int c=10; c++;</pre>
--	Уменьшение на 1	Уменьшим переменную с на 1: <pre>int c=10; c--;</pre>
%	Остаток от деления	Проверим, делится ли переменная а на переменную с без остатка: <pre>int a=10; int c=2; if (a%c==0)     Console.WriteLine("Да"); else     Console.WriteLine("Нет");</pre>
==	Проверка на равенство	Проверим, равны ли а и с: <pre>int a=10; int c=2; if (a==c)     Console.WriteLine("Да"); else     Console.WriteLine("Нет");</pre>
!=	Проверка на различие (не равно)	Проверим, различны ли а и с: <pre>int a=10; int c=2; if (a!=c)     Console.WriteLine("Да"); else     Console.WriteLine("Нет");</pre>

II

СТАНДАРТНЫЙ КУРС



# 3

## Проект № 1 «Стандартные алгоритмы LU-разложения»

### 3.1 Алгоритмы метода Гаусса

Обычный метод Гаусса, осуществляющий LU-разложение матрицы  $A$  [3, 4, 5, 10], заключается в последовательном исключении переменных из уравнений системы

$$Ax = f. \quad (3.1)$$

На первом шаге для исключения первой переменной  $x_1$  из всех уравнений, лежащих в системе ниже первого уравнения, первое уравнение

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = f_1 \quad (3.2)$$

объявляем *ведущим уравнением*. Это возможно только при  $a_{11} \neq 0$ . Тогда, разделив обе части (3.2) на  $a_{11}$ , это ведущее уравнение получим в виде, в котором коэффициент при  $x_1$  окажется равен 1. Заметим, что эта 1 – строгая (т. е. не приближенная) величина. Это действие – деление уравнения на *ведущий элемент* (на первом шаге это  $a_{11} \neq 0$ ) – удобно называть *нормировкой*.

Второе действие заключается в серии вычитаний ведущего уравнения из всех нижележащих уравнений, чтобы исключить из них неизвестную  $x_1$ . Для этого умножаем пронормированное уравнение на  $a_{i1}$  и вычитаем результат из  $i$ -го уравнения системы (3.1),  $i = 2, \dots, n$ . На этом заканчивается первый шаг алгоритма. После первого шага система (3.1) приведена к виду:

$$\left. \begin{aligned} 1 \cdot x_1 + a_{12}^{(1)}x_2 + \dots + a_{1n}^{(1)}x_n &= f_1^{(1)}, \\ a_{22}^{(1)}x_2 + \dots + a_{2n}^{(1)}x_n &= f_2^{(1)}, \\ &\vdots \\ a_{n2}^{(1)}x_2 + \dots + a_{nn}^{(1)}x_n &= f_n^{(1)}. \end{aligned} \right\} \quad (3.3)$$

Это второе действие удобно называть *обновлением активной подсистемы* (активной подматрицы), т. е., той части системы уравнений (матрицы  $A$ ), где ещё будут продолжаться подобного рода действия.

На втором шаге метода Гаусса описанный алгоритм повторяем для переменной  $x_2$ , т. е., берём систему (3.3), объявляем ведущим второе уравнение (для этого нужно иметь  $a_{22}^{(1)} \neq 0$ ) и нормируем в ней второе уравнение. Получаем его в виде

$$1 \cdot x_2 + a_{23}^{(2)} x_3 + \dots + a_{2n}^{(2)} x_n = f_2^{(2)}$$

(верхний индекс в скобках указывает номер шага, после которого получен текущий результат). После этого исключаем переменную  $x_2$  из оставшихся  $n - 2$  уравнений системы (3.3). Таким образом, любой полный шаг алгоритма метода Гаусса состоит из двух действий: сначала нормировка ведущей строки матрицы, потом обновление (серия вычитаний) в активной подматрице.

После  $n - 1$  полных шагов и  $n$ -го неполного шага (поскольку ниже  $n$ -го ведущего уравнения больше нет уравнений) получим две системы линейных алгебраических уравнений

$$Ly = f, \quad \overline{U}x = y, \quad (3.4)$$

эквивалентных исходной системе (3.1), где  $L$  – нижняя треугольная матрица и  $\overline{U}$  верхняя треугольная матрица, на диагонали которой стоят единицы<sup>1</sup>. При этом  $k$ -й столбец матрицы  $L$  (его нетривиальная, т. е., ненулевая часть) запоминает числа для двух действий на  $k$ -м шаге алгоритма, а именно: элемент  $l_{kk}$  является ведущим элементом, производившим нормировку  $k$ -го уравнения, в то время как элементы  $l_{ki}$ ,  $i = k + 1, k + 2, \dots, n$  являются теми коэффициентами, на которые было умножено пронормированное ведущее ( $k$ -е) уравнение, чтобы результатом последующего его вычитания из нижележащих уравнений было исключение из них неизвестного  $x_k$ . Можно говорить, что роль матрицы  $L$  – сохранять «историю» нормировок и вычитаний в процессе исключения неизвестных по методу Гаусса. Роль матрицы  $\overline{U}$  – иная. Матрица  $\overline{U}$  представляет собою тот эквивалентный вид системы (3.1), который она приобретет по завершении этого процесса.

**Определение 3.1.** Определители  $\Delta_i$  подматриц, получаемых оставлением первых  $i$  строк и первых  $i$  столбцов матрицы, называют *главными минорами* матрицы.

**Теорема 3.1.** Если все главные миноры матрицы  $A$  в системе (3.1) отличны от нуля, то процесс Гаусса исключения неизвестных, протекающий

<sup>1</sup> Здесь и далее черта над матрицей означает, что на главной диагонали стоят строгие единицы.

в *прямом направлении*, — начиная от первого неизвестного  $x_1$  и от первого уравнения системы, — эквивалентен разложению  $A = L\bar{U}$ , которое существует и единственno с *нижней* треугольной невырожденной матрицей  $L$  и верхней треугольной матрицей  $\bar{U}$  с единичной диагональю.

**Доказательство.** Докажите теорему 3.1 самостоятельно индукцией по размеру  $n$  матрицы [5]. ■

После разложения вводят правую часть  $f$  данной системы (3.1) и находят вектор  $y$  решения; это называется *прямой подстановкой*:

$$y_i = \left( f_i - \sum_{j=1}^{i-1} l_{ij} y_j \right) l_{ii}^{-1}, \quad i = 1, 2, \dots, n. \quad (3.5)$$

Далее вторая система из (3.4) так же легко решается процедурой *обратной подстановки*:

$$x_i = y_i - \sum_{j=i+1}^n u_{ij} x_j, \quad i = n-1, \dots, 1, \quad x_n = y_n. \quad (3.6)$$

**Замечание 3.1.** Пересчёт элементов вектора  $f$  должен быть отложен, т. е., сначала пересчёт коэффициентов матрицы  $\underline{A}$  должен привести к разложению матрицы  $A$  в произведение матриц  $L$  и  $\bar{U}$  и только затем должна быть введена и соответственно обработана правая часть — вектор  $f$ . При этом вектор  $y$  замещает  $f$  и затем  $x$  замещает  $y$ , — экономия памяти.

### Алгоритм 1. $L\bar{U}$ -разложение матрицы $A$

Для  $k = 1$  до  $n$

Нормируем первую строку матрицы  $A^{(k-1)}$ .

Для  $i = k + 1$  до  $n$

Вычитаем первую строку матрицы  $A^{(k-1)}$ ,  
умноженную на  $a_{ik}^{(k-1)}$ , из  $i$ -й строки.

Здесь  $A^{(k-1)}$  означает активную подматрицу матрицы  $A$  после  $(k-1)$ -го шага метода Гаусса,  $k = 1, 2, \dots, n$ , причём  $A^{(0)} = A$ .

Следующий алгоритм отличается от алгоритма 1 только нормировкой элементов активной подматрицы:

**Алгоритм 2.  $\bar{L}U$ -разложение матрицы  $A$** 

Для  $k = 1$  до  $n - 1$   
Нормируем первый столбец матрицы  $A^{(k-1)}$ .  
Для  $i = k + 1$  до  $n$   
Вычитаем первую строку матрицы  $A^{(k-1)}$ ,  
умноженную на  $a_{ik}^{(k-1)}$ , из  $i$ -й строки.

**Упражнение 3.1.** Измените направление процесса Гаусса на *обратное*.

Докажите, что если процесс Гаусса исключения неизвестных вести от последнего неизвестного  $x_n$  и от последнего уравнения системы, двигаясь вверх по нумерации уравнений и влево по нумерации исключаемых неизвестных, то получим разложение  $A = \bar{U}\bar{L}$  (если нормировать строки) и  $A = \bar{U}L$  (если нормировать столбцы). Докажите соответствующие теоремы – аналоги теоремы 3.1. Для этого измените определение 3.1 главных миноров.

Описанные выше алгоритмы в том виде, в каком они приведены, на практике используются очень редко, т. е., только в том случае, если можно гарантировать, что в результате гауссова исключения на диагонали не появятся нулевые элементы. Однако это можно гарантировать только для матриц специального вида (например, для положительно определенных матриц, см. разд. 4.1), поэтому в общем случае используется метод Гаусса с выбором *главного* (ведущего) элемента.

## 3.2 Выбор ведущего элемента

**Определение 3.2.** Матрицей перестановок  $P$  называют квадратную матрицу, в которой каждая строка, а также каждый столбец, содержит один ненулевой элемент, равный 1.

**Определение 3.3.** Элементарной матрицей перестановок  $P_{ij}$  называют квадратную матрицу, полученную из единичной матрицы перестановкой двух строк:  $i$ -й и  $j$ -й, либо перестановкой двух столбцов:  $i$ -го и  $j$ -го (оба варианта перестановок дают один и тот же результат).

**Упражнение 3.2.** Докажите справедливость следующих утверждений:

1. Произведение  $P_{ij}A$  производит в  $A$  перестановку  $i$ -й и  $j$ -й строк.

2. Произведение  $AP_{ij}$  производит в  $A$  перестановку  $i$ -го и  $j$ -го столбцов.
3.  $P_{ij}P_{ij} = I$  – свойство *идемпотентности* матриц  $P_{ij}$ .
4. Любая матрица перестановок  $P$  может быть разложена в произведение элементарных матриц перестановок.
5.  $P^{-1} = P^T$  – свойство *ортогональности* матриц перестановок  $P$ .

**Теорема 3.2.** Если  $\det A \neq 0$ , то существуют матрицы перестановок  $P$  и  $Q$  такие, что все угловые (т. е., главные) миноры матрицы  $PA$ , равно как и матриц  $AP$  и  $APQ$ , отличны от нуля.

**Доказательство.** Докажите индукцией по размеру матрицы  $A$  [5]. ■

**Следствие 3.1.** Если  $\det A \neq 0$ , то существуют матрицы перестановок  $P$  и  $Q$  такие, что следующие варианты разложения матрицы  $A$  существуют и единственны:  $PA = LU$ ,  $PA = \overline{L}U$ ,  $AP = \overline{L}U$ ,  $AP = LU$ ,  $PAQ = \overline{L}U$ ,  $PAQ = \overline{L}U$ . ■

Отсюда видны *три стратегии выбора главного (ведущего) элемента*: по столбцу, по строке и по активной подматрице.

**Стратегия I.** Первая стратегия (*по столбцу*) подразумевает, что в качестве главного на  $k$ -м шаге метода Гаусса выбирается максимальный по модулю элемент первого столбца активной подматрицы. Затем этот элемент меняется местами с диагональным элементом, что соответствует перестановке строк матрицы  $A^{(k-1)}$  и элементов вектора  $f^{(k-1)}$ . На самом деле строки матрицы  $A^{(k-1)}$  и элементы вектора  $f^{(k-1)}$  остаются на своих местах, а переставляются только элементы дополнительного вектора, в котором хранятся номера строк исходной матрицы, соответствующие номерам строк матрицы, т. е., элементы так называемого *вектора перестановок*. Все обращения к элементам матриц  $L$ ,  $U$  и вектора  $f$  осуществляются через вектор перестановок.

**Стратегия II.** Следующая стратегия (*по строке*) заключается в выборе в качестве главного элемента максимального по модулю элемента первой строки активной подматрицы. Затем этот элемент обменивается местами с диагональным, что соответствует перестановке столбцов матрицы  $A^{(k-1)}$  и элементов вектора  $x$ . Как и в предыдущем случае, реально обмениваются

только элементы дополнительного вектора, в котором фиксируются перестановки столбцов матрицы  $A$ . Доступ к элементам матриц  $L$ ,  $U$  и вектора  $x$  осуществляется с использованием этого вектора.

**Стратегия III.** Последняя стратегия выбора главного элемента (*по активной подматрице*) объединяет две первые стратегии. Здесь в качестве главного выбирается максимальный по модулю элемент активной подматрицы. В общем случае, чтобы поставить этот элемент на место диагонального, требуется обменивать столбцы и строки матрицы  $A^{(k-1)}$ , что связано с введением двух дополнительных векторов: в одном хранятся перестановки столбцов, а в другом — перестановки строк матрицы  $A$ .

Приведённые выше алгоритмы  $LU$ -разложения с учётом выбора главного элемента преобразуются к одному из следующих вариантов.

### Алгоритм 3. $\overline{LU}$ -разложение по методу Гаусса с выбором главного элемента

```

Для  $k = 1$  до  $n$ 
    Выбираем главный элемент в  $A^{(k-1)}$ .
    Нормируем первую строку матрицы  $A^{(k-1)}$ .
    Для  $i = k + 1$  до  $n$ 
        Вычитаем первую строку матрицы  $A^{(k-1)}$ ,
        умноженную на  $a_{ik}^{(k-1)}$ , из  $i$ -й строки.

```

### Алгоритм 4. $\overline{LU}$ -разложение по методу Гаусса с выбором главного элемента

```

Для  $k = 1$  до  $n - 1$ 
    Выбираем главный элемент в  $A^{(k-1)}$ .
    Нормируем первый столбец матрицы  $A^{(k-1)}$ .
    Для  $i = k + 1$  до  $n$ 
        Вычитаем первую строку матрицы  $A^{(k-1)}$ 
        умноженную на  $a_{ik}^{(k-1)}$  из  $i$ -й строки.

```

Вышеприведённые алгоритмы называют *исключением по столбцам*, так как они исключают  $x_k$  из всей поддиагональной части  $k$ -го столбца.

**Замечание 3.2.** Во всех алгоритмах должно быть выполнено требование к реализации: все действия должны выполняться в одном и том же массиве чисел. Например, в Алгоритме 1 сначала  $A^{(0)} = A$ , а в конце на месте этой матрицы получаются нетривиальные элементы матриц  $L$  и  $\bar{U}$ .

**Замечание 3.3.** Под выбором главного элемента здесь и далее понимается любая из описанных выше трёх стратегий, которая применима.

**Замечание 3.4.** При  $UL$ -разложении матрицы  $A$  все действия выполняются аналогично, но в обратном порядке нумерации строк и столбцов: снизу-вверх и справа-налево.

**Замечание 3.5.** В описаниях алгоритмов упоминаются элементы матрицы  $A$ . Естественно, речь идёт о текущих, а не об исходных значениях элементов, занимающих указанные позиции матрицы  $A$ . Это связано с выполнением требования к реализации (см. Замечание 3.2).

Наряду с гауссовым исключением по столбцам, представленным выше, возможно проводить гауссово исключение по строкам. Это такая разновидность метода Гаусса, в которой на каждом шаге исключения изменяется одна строка — первая строка активной подматрицы.

Рассмотрим ещё один вариант гауссова исключения — гауссово исключение по строкам с выбором главного элемента по строке.

Выполняем  $i$ -й шаг, т. е., работаем с  $i$ -й строкой матрицы. В ней ещё не было ни одного исключения неизвестных. Первое действие: из  $i$ -й строки вычитаем первую, умноженную на  $a_{i1}$ ; затем из  $i$ -й строки вычитаем вторую, умноженную на  $a_{i2}$ , и так далее; в завершение этой серии вычитаний из  $i$ -й строки вычитаем  $(i-1)$ -ю, умноженную на  $a_{i(i-1)}$ . Второе действие: отыскиваем главный элемент в  $i$ -й строке и осуществляем (если надо) перестановку столбцов. Третье действие:  $i$ -ю строку нормируем (делим на ведущий элемент). Повторяя этот алгоритм  $n$  раз, получим  $\bar{U}$ -разложение матрицы  $AP$ . При  $i = 1$  шаг, очевидно, неполный, т. е., без вычитаний.

Таким образом, отличие гауссова исключения по строкам от гауссова исключения по столбцам сводится в алгоритме к изменению порядка действий: сначала серия вычитаний, а затем — нормировка. Такая возможность (для варианта  $A = \bar{U}$ ) представлена в следующем алгоритме.

**Алгоритм 5.  $L\bar{U}$ -разложение по методу Гаусса (по строкам)**

```

Для  $k = 1$  до  $n$ 
    Для  $i = 1$  до  $k - 1$ 
        Вычитаем  $i$ -ю строку матрицы  $A$ ,
        умноженную на  $a_{ki}$ , из  $k$ -й строки.
    Выбираем главный элемент в  $k$ -й строке.
    Нормируем  $k$ -ю строку матрицы  $A$ .

```

### 3.3 Компактные схемы

Следующей разновидностью метода Гаусса являются *компактные схемы*. Первая называется *компактной схемой Краута*, а вторую мы будем называть *компактной схемой Краута «строка за строкой»*. В схеме Краута на каждом шаге исключения изменяются только первый столбец и первая строка активной подматрицы. В схеме «строка за строкой» на  $k$ -м шаге изменяется только  $k$ -я строка матрицы  $A$ .

Выведем формулы схемы Краута для  $k$ -го шага. Предположим, что уже сделаны первые  $k - 1$  шагов, т. е. определены  $k - 1$  столбец матрицы  $L$  и  $k - 1$  строка матрицы  $\bar{U}$ . Из соотношения  $A = L\bar{U}$  для  $(i, j)$ -го элемента имеем

$$a_{ij} = \sum_{p=1}^n l_{ip} u_{pj}. \quad (3.7)$$

В силу треугольности матриц  $L$  и  $\bar{U}$  при  $p > i$  имеем  $l_{ip} = 0$  и при  $p > j$  имеем  $u_{pj} = 0$ . Тогда с учётом того, что  $u_{kk} = 1$ , для  $k$ -го столбца матрицы  $A$  находим

$$a_{ik} = l_{ik} + \sum_{p=1}^{k-1} l_{ip} u_{pk}, \quad i \geq k. \quad (3.8)$$

Из (3.8) следует

$$l_{ik} = a_{ik} - \sum_{p=1}^{k-1} l_{ip} u_{pk}, \quad i \geq k. \quad (3.9)$$

Следовательно,  $k$ -й столбец матрицы  $L$  становится известен. Теперь из (3.7) для  $k$ -й строки матрицы  $A$  имеем

$$a_{kj} = l_{kk}u_{kj} + \sum_{p=1}^{k-1} l_{kp}u_{pj}, \quad j > k. \quad (3.10)$$

Из (3.10) находим

$$u_{kj} = \left( a_{kj} - \sum_{p=1}^{k-1} l_{kp}u_{pj} \right) / l_{kk}, \quad j > k. \quad (3.11)$$

Таким образом, (3.11) даёт способ нахождения  $k$ -й строки матрицы  $\bar{U}$ . В результате, зная первые  $k - 1$  столбцов матрицы  $L$  и  $k - 1$  строк матрицы  $U$ , мы можем по формулам (3.9) и (3.11) определить  $k$ -й столбец матрицы  $L$  и затем  $k$ -ю строку матрицы  $\bar{U}$ . Первый столбец матрицы  $L$  определяется равенствами

$$l_{i1} = a_{i1}, \quad i = 1, 2, \dots, n.$$

Это следует из (3.9) и того, что первым столбцом матрицы  $\bar{U}$  является первый координатный вектор  $e_1$ . Здесь, как обычно, предполагаем, что если нижний предел суммирования меньше верхнего, то значение суммы равно нулю. После этого в первом столбце выбираем главный элемент. Затем по формулам

$$u_{1j} = a_{1j} / l_{1j}, \quad j = 2, 3, \dots, n$$

вычисляем первую строку матрицы  $\bar{U}$ . Повторяя указанную последовательность действий  $n$  раз, с помощью формул (3.9) и (3.11) получаем  $\bar{LU}$ -разложение матрицы  $A$ .

#### Алгоритм 6. $\bar{LU}$ -разложение по компактной схеме Краута

Для  $k = 1$  до  $n$

По формуле (3.9) вычисляем  $k$ -й столбец матрицы  $L$ .

Выбираем среди элементов  $k$ -го столбца главный элемент.

По формуле (3.11) вычисляем  $k$ -ю строку матрицы  $\bar{U}$ .

Чтобы получить метод Краута, дающий  $\bar{LU}$ -разложение с выбором главного элемента по строке, достаточно поменять местами формулы (3.9) и (3.11), а

также последовательность вычисления столбцов матрицы  $\bar{L}$  и строк матрицы  $\bar{U}$ . Таким образом, на  $k$ -м шаге сначала по формуле

$$u_{kj} = a_{kj} - \sum_{p=1}^{k-1} l_{kp} u_{pj}, \quad j \geq k, \quad (3.12)$$

вычисляется строка матрицы  $\bar{U}$ . Затем в этой строке выбирается главный элемент и находится столбец матрицы  $\bar{L}$  по следующей формуле:

$$l_{ik} = \left( a_{ik} - \sum_{p=1}^{k-1} l_{ip} u_{pk} \right) / u_{kk}, \quad i \geq k. \quad (3.13)$$

**Упражнение 3.3.** Выведите расчетные формулы компактных схем для любого из альтернативных вариантов разложения:  $A = \bar{U}\bar{L}$  или  $A = \bar{U}L$ . Что изменяется? Дайте ответы в форме условных схем алгоритмов, представленных непосредственно до и после этого упражнения.

### Алгоритм 7. $\bar{LU}$ -разложение по компактной схеме Краута

Для  $k = 1$  до  $n$

По формуле (3.12) вычисляем  $k$ -ю строку матрицы  $\bar{U}$ .

Выбираем среди элементов  $k$ -й строки главный элемент.

По формуле (3.13) вычисляем  $k$ -й столбец матрицы  $\bar{L}$ .

Компактная схема «строка за строкой», дающая  $\bar{LU}$ -разложение матрицы  $A$ , использует те же самые формулы (3.9) и (3.11). Меняется только последовательность вычисления элементов матрицы  $L$ . Рассмотрим подробнее.

Пусть уже обработаны по этому методу первые  $k - 1$  строк матрицы  $\bar{A}$ . Следовательно, мы имеем  $k - 1$  строку матрицы  $\bar{L}$  и  $k - 1$  строку матрицы  $\bar{U}$ . Далее по формулам (3.9) вычисляем ненулевые элементы  $k$ -й строки матрицы  $\bar{L}$ . По формулам (3.11) без деления на диагональный элемент  $\bar{l}_{kk}$  вычисляем ненулевые элементы  $k$ -й строки матрицы  $\bar{U}$ . Затем среди вновь вычисленных элементов, от диагонального до  $n$ -го, определяем главный элемент, меняем его местами с диагональным и делим элементы  $k$ -й строки матрицы  $\bar{U}$  на этот элемент. В результате получаем требуемое разложение.

**Алгоритм 8.**  $\bar{LU}$ -разложение по компактной схеме  
«строка за строкой»

Для  $k = 1$  до  $n$

По формуле (3.9) вычисляем элементы  $k$ -й строки матрицы  $L$ .

По формуле (3.11) без деления на диагональный элемент  $l_{kk}$ , вычисляем  $k$ -ю строку матрицы  $\bar{U}$ .

Среди элементов  $k$ -й строки (от диагонального до  $n$ -го) определяем главный элемент.

Делим на главный элемент  $k$ -ю строку матрицы  $\bar{U}$ .

## 3.4 Алгоритмы метода Жордана

К последней группе методов исключения относятся алгоритмы метода Жордана. Эти алгоритмы используют те же самые формулы, что и обычный метод Гаусса, но в отличие от него на  $k$ -м шаге метода Жордана пересчитывают все строки матрицы  $A$ , а не только строки, находящиеся ниже ведущей строки. Это означает полное исключение  $i$ -й переменной из всех уравнений, кроме  $i$ -го. Таким образом, метод Жордана формально даёт решение системы линейных алгебраических уравнений за один проход.

**Теорема 3.3.** Выполнение действий полного исключения в том же массиве, где первоначально располагалась матрица  $A$ , даёт то же самое разложение  $A = \bar{LU}$ , что и метод Гаусса, но существенно в другом виде, а именно: матрица  $L$  получается, как и в методе Гаусса, но матрица  $\bar{U}$  оказывается полученной не в «чистом виде», а в виде обратной матрицы  $\bar{U}^{-1}$  и с той особенностью, что все знаки элементов матрицы  $\bar{U}^{-1}$  выше диагонали имеют неправильные (противоположные) знаки.

**Доказательство.** Нужно воспользоваться определениями элементарных матриц специального вида [11] и их свойствами.

Приведём эти определения и свойства и затем продолжим доказательство.

**Определение 3.4.** Элементарная матрица  $E$  есть любая матрица вида  $E = I + B$ , где  $\text{rank } B = 1$ .

**Упражнение 3.4.** Докажите, что  $E = I + xy^T$ , где  $x$  и  $y$  – некоторые векторы.

**Определение 3.5.** Введём следующие *специальные элементарные матрицы*:

$D_k$  – диагональная  $k$ -матрица. Имеет единичную диагональ, кроме  $k$ -го элемента, который не тривиален, т. е., не равен нулю или единице.

$L_k^C$  – столбцово-элементарная нижняя треугольная  $k$ -матрица. Имеет единичную диагональ и нетривиальные элементы только в  $k$ -м столбце.

$L_k^R$  – строчно-элементарная нижняя треугольная  $k$ -матрица. Имеет единичную диагональ и нетривиальные элементы только в  $k$ -й строке.

$U_k^C$  – столбцово-элементарная верхняя треугольная  $k$ -матрица. Имеет единичную диагональ и нетривиальные элементы только в  $k$ -м столбце.

$U_k^R$  – строчно-элементарная верхняя треугольная  $k$ -матрица. Имеет единичную диагональ и нетривиальные элементы только в  $k$ -й строке.

$T_k^C$  – полно-столбцово-элементарная верхняя треугольная  $k$ -матрица. Она содержит единичную диагональ и нетривиальные элементы только в  $k$ -м столбце.

$T_k^R$  – полно-строчно-элементарная верхняя треугольная  $k$ -матрица. Имеет единичную диагональ и нетривиальные элементы только в  $k$ -й строке.

**Упражнение 3.5.** Докажите свойства этих элементарных матриц, приведённые в табл. 3.1.

Продолжим доказательство теоремы 3.3, прерванное на стр. 97.

Приведение данной матрицы  $A$  к единичной матрицы, составляющее суть исключения по методу Гаусса-Жордана, запишем в терминах операций с введенными специальными элементарными матрицами:

$$A^{(n+1)} = (T_n^C)^{-1} D_n^{-1} \cdots (T_2^C)^{-1} D_2^{-1} (T_1^C)^{-1} D_1^{-1} A = I. \quad (3.14)$$

К результату (3.14) приводит алгоритм Гаусса-Жордана, показанный ниже на с. 99 для случая  $LU$ -разложения матрицы  $A$ .

Таблица 3.1. Свойства специальных элементарных матриц

Коммутативность в операции умножения	
$L_i^C D_j = D_j L_i^C, \quad i > j$	$U_i^C D_j = D_j U_i^C, \quad i < j$
$L_i^R D_j = D_j L_i^R, \quad i < j$	$U_i^R D_j = D_j U_i^R, \quad i > j$
$L_i^C U_j^C = U_j^C L_i^C, \quad i \geq j$	$L_i^R U_j^R = U_j^R L_i^R, \quad i \leq j$
$L_i^C U_i^C = U_i^C L_i^C = T_i^C$	$L_i^R U_i^R = U_i^R L_i^R = T_i^R$
Операция обращения матриц	
$D_i^{-1}$ получается из $D_i$ заменой нетривиального элемента $d_{ii}$ на $d_{ii}^{-1}$ с сохранением знака этого элемента	$E^{-1}$ , где $E \in \{L_k^C, L_k^R, U_k^C, U_k^R, T_k^C, T_k^R\}$ получается из $E$ заменой знаков нетривиальных элементов на противоположные
Применимость правила суперпозиции вместо перемножения матриц	
$L_i^C L_j^C L_k^C, \quad i < j < k$	$L_i^R L_j^R L_k^R, \quad i < j < k$
$U_i^C U_j^C U_k^C, \quad i > j > k$	$U_i^R U_j^R U_k^R, \quad i > j > k$
$L = D_1 L_1^C D_2 L_2^C \cdots D_{n-1} L_{n-1}^C D_n L_n^C$	$L = L_1^R D_1 L_2^R D_2 \cdots L_{n-1}^R D_{n-1} L_n^R D_n$
$U = D_n U_n^C D_{n-1} U_{n-1}^C \cdots D_2 U_2^C D_1 U_1^C$	$U = U_n^R D_n U_{n-1}^R D_{n-1} \cdots U_2^R D_2 U_1^R D_1$

### Алгоритм 9. $\bar{LU}$ -разложение по методу Гаусса-Жордана

Начальное значение:  $A^{(1)} = A$ .

Для  $k = 1$  до  $n$  выполнять

$$A^{(k+1)} = (T_k^C)^{-1} D_k^{-1} A^{(k)},$$

где элементы

$$D_k^{-1}(k, k) = 1/A^{(k)}(k, k),$$

$$T_k^C(i, k) = A^{(k)}(k, k)(i, k), \quad i = 1, 2, \dots, n, \quad i \neq k$$

суть множители для нормировки и вычитаний, соответственно.

Множители, возникающие в этом алгоритме, образуют так называемую таблицу множителей. По существу, это матрица, которая займёт место исходной матрицы  $A$  по окончании всего алгоритма:

$$\begin{bmatrix} D_1^{-1}(1, 1) & T_1^C(1, 2) & T_1^C(1, 3) & \cdots \\ T_1^C(2, 1) & D_1^{-1}(2, 2) & T_1^C(2, 3) & \cdots \\ T_1^C(3, 1) & T_1^C(3, 2) & D_1^{-1}(3, 3) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \quad (3.15)$$

Воспользуемся свойством в четвёртой строке табл. 3.1,  $T_i^C = L_i^C U_i^C$  в его инверсной форме  $(T_i^C)^{-1} = (U_i^C)^{-1} (L_i^C)^{-1}$ , и подставим его в (3.14), а также свойствами коммутативности из этой таблицы. Это даёт возможность перегруппировать сомножители в (3.14) следующим образом:

$$[(U_n^C)^{-1} \cdots (U_2^C)^{-1} (U_1^C)^{-1}] \cdot [(L_n^C)^{-1} D_n^{-1} \cdots (L_2^C)^{-1} D_2^{-1} (L_1^C)^{-1} D_1^{-1}] A = I.$$

Второй сомножитель в квадратных скобках совпадает с матрицей  $L^{-1}$  для  $\bar{LU}$ -разложения матрицы  $A$ . Первый сомножитель в квадратных скобках есть матрица  $U^{-1}$  для этого разложения. Правило суперпозиции, согласно табл. 3.1, действует для первого сомножителя (в нём индексы матриц убывают слева-направо) и не действует для второго сомножителя. Однако для  $L = D_1 L_1^C D_2 L_2^C \cdots D_{n-1} L_{n-1}^C D_n L_n^C$  правило суперпозиции действует. Суперпозиция, т. е., постановка элементов матриц на принадлежащие им позиции, реализуется автоматически по ходу алгоритма. Поэтому в нижней треугольной части матрицы (3.15) (кроме диагонали) образуется матрица  $L$ , диагональные элементы матрицы  $L$  представлены на диагонали, но в инверсном виде (там – обратные значения этих элементов), а выше диагонали расположены элементы той матрицы, для которой действует правило суперпозиции, т. е., элементы матрицы  $U^{-1}$ . В работе алгоритма перемены знаков у этих элементов не совершились (что надо делать при обращении элементарных матриц  $E$ , согласно табл. 3.1), поэтому эти знаки – неверные.

Чтобы выполнить разложение по методу Жордана, надо воспользоваться следующим алгоритмом. На первом шаге в активной подматрице  $A^0 = A$  выбирается главный элемент. Затем первая строка нормируется, домножается на  $a_{i1}$  и вычитается из  $i$ -й строки,  $i = 2, 3, \dots, n$ . На втором шаге главный элемент определяется среди элементов активной подматрицы  $A^{(1)}$ . Потом вторая строка нормируется и после домножения на  $a_{i2}$  вычитается из  $i$ -й, где  $i = 1, 3, \dots, n$ . В общем случае на  $k$ -м шаге в подматрице  $A^{(k-1)}$  выбирается главный элемент. Затем  $k$ -я строка нормируется, домножается на  $a_{ik}$  и вычитается из  $i$ -й, где  $i = 1, \dots, k-1, k+1, \dots, n$ . В результате, чтобы получить требуемое разложение, остаётся поменять знак на противоположный у всех элементов, лежащих выше главной диагонали.

**Замечание 3.6.** Термин « $\bar{LU}^{-1}$ -разложение», который мы используем здесь повсюду для краткости в применении к методу Жордана, не должен вводить в заблуждение. Он самом деле отыскивает  $\bar{LU}$ -разложение матрицы

$A$ , но при его выполнении в одном и том же массиве даёт вместо матрицы  $\bar{U}$  обратную матрицу  $\bar{U}^{-1}$ , причём в следующем виде: единицы главной диагонали матрицы  $\bar{U}^{-1}$  не хранятся, а все другие элементы этой матрицы получаются с противоположными знаками.

### Алгоритм 10. « $\bar{L}\bar{U}^{-1}$ -разложение» $A = \bar{L}\bar{U}$ по методу Жордана

Для  $k = 1$  до  $n$

Выбираем главный элемент в  $A^{(k-1)}$ .

Нормируем первую строку матрицы  $A^{(k-1)}$ .

Для  $i = 1$  до  $k - 1$

Вычитаем первую строку матрицы  $A^{(k-1)}$ , умноженную на  $a_{ik}^{(k-1)}$ , из  $i$ -й строки.

Для  $i = k + 1$  до  $n$

Вычитаем первую строку матрицы  $A^{(k-1)}$ , умноженную на  $a_{ik}^{(k-1)}$ , из  $i$ -й строки.

Для  $i = 1$  до  $n$

Для  $j = i + 1$  до  $n$

$$a_{ij} = -a_{ij}$$

**Упражнение 3.6.** Объясните, как следует понимать словосочетание « $\bar{L}^{-1}\bar{U}$ -разложение» Жордана. Докажите, что в этом случае выполняется разложение  $A = \bar{L}\bar{U}$ , но матрица  $\bar{L}$  получается в виде обратной матрицы с неправильными знаками её внедиагональных элементов.

**Замечание 3.7.** Чтобы сэкономить процессорное время, целесообразно везде пользоваться обратными величинами ведущих элементов, как это сделано на диагонали в таблице множителей (3.15).

## 3.5 Вычисление обратной матрицы

Есть два способа вычисления обратной матрицы  $A^{-1}$ : через решение системы  $Ax = f$  с различными правыми частями  $f$  и непосредственно через разложение матрицы  $A$  в произведение треугольных матриц. В первом способе правая часть  $f$  последовательно пробегает значения столбцов  $e_i$  единичной матрицы  $I$ , при этом для каждой из них найденное решение  $x$  системы  $Ax = f$

образует  $i$ -й столбец искомой матрицы  $A^{-1}$ . Это, очевидно, соответствует решению матричного уравнения  $AX = I$ , так как  $X = A^{-1}$ .

Второй способ основан на том, что если  $A = LU$ , то  $A^{-1} = \overline{U}^{-1}L^{-1}$ . Это называют *элиминативной формой обратной матрицы* [11], так как здесь  $A^{-1}$  находят непосредственно по разложению  $A = LU$ , которое само по себе эквивалентно процедуре гауссова исключения (*elimination*) неизвестных. Рассмотрим этот способ и характеризуем особенности его программной реализации более подробно. Для численной иллюстрации рассмотрим следующий пример.

**Пример 3.1.** Пусть для данной матрицы  $A$  найдено  $A = LU$ :

$$A = \begin{bmatrix} 2 & 4 & -4 & 6 \\ 1 & 4 & 2 & 1 \\ 3 & 8 & 1 & 1 \\ 2 & 5 & 0 & 5 \end{bmatrix}, L = \begin{bmatrix} 2 & & & \\ 1 & 2 & & \\ 3 & 2 & 3 & \\ 2 & 1 & 2 & 4 \end{bmatrix}, \overline{U} = \begin{bmatrix} 1 & 2 & -2 & 3 \\ & 1 & 2 & -1 \\ & & 1 & -2 \\ & & & 1 \end{bmatrix}.$$

Известная особенность реализации такого разложения заключается в том, что результат разложения замещает исходную матрицу, т. е., имеем

исходный массив  $\Rightarrow$  результирующий массив

$$\begin{bmatrix} 2 & 4 & -4 & 6 \\ 1 & 4 & 2 & 1 \\ 3 & 8 & 1 & 1 \\ 2 & 5 & 0 & 5 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 2 & -2 & 3 \\ 1 & 2 & 2 & -1 \\ 3 & 2 & 3 & -2 \\ 2 & 1 & 2 & 4 \end{bmatrix}. \quad (3.16)$$

Следовательно, до начала вычисления обратной матрицы  $A^{-1}$  в наличии имеем две матрицы: матрицу  $L$  – в нижней треугольной части массива вместе с диагональю, матрицу  $\overline{U}$  – в верхней треугольной части массива без единичной (известной по умолчанию) диагонали. Запишем  $A^{-1} = \overline{U}^{-1} \times L^{-1}$ , где символ  $\times$  обозначает процедуру перемножения треугольных матриц  $\overline{U}^{-1}$  и  $L^{-1}$  в указанном порядке. Тем самым отмечаем, что это должна быть специальная, а не общая процедура, экономящая время вычислений за счёт исключения операций умножения на заведомо нулевые элементы сомножителей  $\overline{U}^{-1}$  и  $L^{-1}$ .

Сомножители  $\overline{U}^{-1}$  и  $L^{-1}$  нужно вычислять также по специальным процедурам, для которых исходные данные  $\overline{U}$  и  $L$  берутся из массива, названного в выражении (3.16) *результатом массивом* после факторизации  $A = LU$ . Результаты  $\overline{U}^{-1}$  и  $L^{-1}$  работы этих процедур записываются в этот же результирующий массив.

Вывод алгоритмов процедур для  $L^{-1}$  и для  $\bar{U}^{-1}$  основан на свойствах элементарных треугольных матриц, в частности, на свойстве «суперпозиции вместо перемножения». Для  $L$  это свойство означает, что произведение  $L = L_1 L_2 L_3 L_4$  может быть получено не перемножением элементарных матриц  $L_1, L_2, L_3, L_4$ , а суперпозицией (постановкой на свои позиции) нетривиальных столбцов элементарных матриц-сомножителей:

$$\begin{bmatrix} L \\ 2 & 1 & 2 & 3 & 2 & 1 & 2 & 4 \end{bmatrix} = \begin{bmatrix} L_1 \\ 2 & 1 & 1 & 3 & 2 & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} L_2 \\ 1 & 2 & 1 & 2 & 1 & 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} L_3 \\ 1 & 1 & 3 & 2 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} L_4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 4 \end{bmatrix}.$$

Согласно правилу обращения произведения матриц,  $L^{-1}$  найдём как результат перемножения следующих обратных матриц:

$$\begin{bmatrix} L_4^{-1} \\ 1 & 1 & 1 & 4 \end{bmatrix}^{-1} \times \begin{bmatrix} L_3^{-1} \\ 1 & 1 & 3 & 2 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} L_2^{-1} \\ 1 & 2 & 2 & 1 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} L_1^{-1} \\ 2 & 1 & 3 & 2 & 1 \end{bmatrix}^{-1}.$$

Так как индексы у элементарных нижнетреугольных матриц здесь слева направо уже не возрастают, а убывают, операция перемножения  $\times$  матриц не может быть заменена суперпозицией. В программной реализации этот символ  $\times$  должен соответствовать некоторой специальной вычислительной процедуре. Все исходные данные для этой процедуры уже предоставлены полученным разложением (3.16). Действительно, инверсию элементарных матриц, показанных в последнем выражении, получают в самой процедуре применением простых операций: сначала диагональный элемент из нетривиального столбца каждой элементарной матрицы заменяют на обратный по величине; затем полученное число берут с противоположным знаком и умножают на каждый поддиагональный элемент. Эти действия соответствуют указанному выражению, представленному в следующем виде:

$$\begin{bmatrix} L_4^{-1} \\ 1 & 1 & 1 & 1/4 \end{bmatrix} \times \begin{bmatrix} L_3^{-1} \\ 1 & 1 & 1/3 & -2/3 & 1 \end{bmatrix} \times \begin{bmatrix} L_2^{-1} \\ 1 & 1/2 & -2/2 & -1/2 & 1 \end{bmatrix} \times \begin{bmatrix} L_1^{-1} \\ 1/2 & -1/2 & -3/2 & -2/2 & 1 \end{bmatrix}.$$

В действительности это означает, что сначала – на этапе ❶ – результирующий массив из (3.16) пересчитывают по указанным правилам, чтобы найти  $L^{-1}$ , приводя этот массив к следующему стартовому виду:

$$\left[ \begin{array}{cccc} 2 & 2 & -2 & 3 \\ 1 & 2 & 2 & -1 \\ 3 & 2 & 3 & -2 \\ 2 & 1 & 2 & 4 \end{array} \right] \xrightarrow{\text{❶}} \left[ \begin{array}{cccc} 1/2 & 2 & -2 & 3 \\ -1/2 & 1/2 & 2 & -1 \\ -3/2 & -2/2 & 1/3 & -2 \\ -2/2 & -1/2 & -2/3 & 1/4 \end{array} \right]. \quad (3.17)$$

Чтобы понять, как в этом массиве должна работать процедура вычисления матрицы  $L^{-1}$ , рассмотрим произведение матриц перед выражением (3.17) и формально будем перемножать матрицы  $L_4^{-1}, L_3^{-1}, L_2^{-1}, L_1^{-1}$  справа налево, т. е., вычислим  $L_4^{-1}(L_3^{-1}(L_2^{-1}L_1^{-1}))$ . Процесс такого поэтапного перемножения отразим в табл. 3.2.

Из табл. 3.2 видно, что после получения (3.17), т. е., на этапе ❷, пересчитывают только элементы  $a_{21}, a_{31}$  и  $a_{41}$ . В данном случае имеем

$$\left[ \begin{array}{cccc} 1/2 & 2 & -2 & 3 \\ -1/2 & 1/2 & 2 & -1 \\ -3/2 & -2/2 & 1/3 & -2 \\ -2/2 & -1/2 & -2/3 & 1/4 \end{array} \right] \xrightarrow{\text{❷}} \left[ \begin{array}{cccc} 1/2 & 2 & -2 & 3 \\ -1/4 & 1/2 & 2 & -1 \\ -1 & -1 & 1/3 & -2 \\ -3/4 & -1/2 & -2/3 & 1/4 \end{array} \right].$$

Далее видно, что на этапе ❸ пересчитывают только  $a_{31}, a_{41}, a_{32}$  и  $a_{42}$ , т. е.,

$$\left[ \begin{array}{cccc} 1/2 & 2 & -2 & 3 \\ -1/4 & 1/2 & 2 & -1 \\ -1 & -1 & 1/3 & -2 \\ -3/4 & -1/2 & -2/3 & 1/4 \end{array} \right] \xrightarrow{\text{❸}} \left[ \begin{array}{cccc} 1/2 & 2 & -2 & 3 \\ -1/4 & 1/2 & 2 & -1 \\ -1/3 & -1/3 & 1/3 & -2 \\ -1/12 & 1/6 & -2/3 & 1/4 \end{array} \right],$$

и на этапе ❹ пересчитываются только элементы  $a_{41}, a_{42}$  и  $a_{43}$ , т. е.,

$$\left[ \begin{array}{cccc} 1/2 & 2 & -2 & 3 \\ -1/4 & 1/2 & 2 & -1 \\ -1/3 & -1/3 & 1/3 & -2 \\ -1/12 & 1/6 & -2/3 & 1/4 \end{array} \right] \xrightarrow{\text{❹}} \left[ \begin{array}{cccc} 1/2 & 2 & -2 & 3 \\ -1/4 & 1/2 & 2 & -1 \\ -1/3 & -1/3 & 1/3 & -2 \\ -1/48 & 1/24 & -1/6 & 1/4 \end{array} \right].$$

Построение алгоритма вычисления матрицы  $\overline{U}^{-1}$  проводится аналогично. Для  $\overline{U}$  свойство суперпозиции означает, что произведение  $\overline{U} = \overline{U}_4 \overline{U}_3 \overline{U}_2 \overline{U}_1$  может быть получено не перемножением элементарных матриц

Таблица 3.2. Поэтапное перемножение  $L_4^{-1}(L_3^{-1}(L_2^{-1}L_1^{-1}))$ 

$$\begin{array}{c}
 \left[ \begin{array}{cccc} & L_4^{-1} & & \\ 1 & 1 & & \\ & 1 & & \\ & & 1/4 & \end{array} \right] \times \left[ \begin{array}{cccc} & L_3^{-1} & & \\ 1 & 1 & & \\ & 1/3 & & \\ & -2/3 & 1 & \end{array} \right] \times \left[ \begin{array}{cccc} & L_2^{-1} & & \\ 1 & & & \\ & 1/2 & & \\ & -2/2 & 1 & \\ & -1/2 & & 1 \end{array} \right] \times \left[ \begin{array}{cccc} & L_1^{-1} & & \\ 1/2 & & & \\ -1/2 & 1 & & \\ -3/2 & & 1 & \\ -2/2 & & & 1 \end{array} \right] \\
 \\
 \left[ \begin{array}{cccc} & L_3^{-1} & & (L_2^{-1}L_1^{-1}) \\ 1 & 1 & & \\ & 1/3 & & \\ & -2/3 & 1 & \end{array} \right] \times \left[ \begin{array}{cccc} 1/2 & & & \\ -1/4 & 1/2 & & \\ -1 & -1 & 1 & \\ -3/4 & -1/2 & & 1 \end{array} \right] \Leftarrow \textcircled{2} \\
 \\
 \left[ \begin{array}{cccc} & L_4^{-1} & & (L_3^{-1}(L_2^{-1}L_1^{-1})) \\ 1 & 1 & & \\ & 1 & & \\ & & 1/4 & \end{array} \right] \times \left[ \begin{array}{cccc} 1/2 & & & \\ -1/4 & 1/2 & & \\ -1/3 & -1/3 & 1/3 & \\ -1/12 & 1/6 & -2/3 & 1 \end{array} \right] \Leftarrow \textcircled{3} \\
 \\
 \left[ \begin{array}{cccc} L_4^{-1}(L_3^{-1}(L_2^{-1}L_1^{-1})) & & & \\ 1/2 & & & \\ -1/4 & 1/2 & & \\ -1/3 & -1/3 & 1/3 & \\ -1/48 & 1/24 & -1/6 & 1/4 \end{array} \right] \Leftarrow \textcircled{4}
 \end{array}$$

$\overline{U}_4, \overline{U}_3, \overline{U}_2, \overline{U}_1$ , а суперпозицией (постановкой на свои позиции) нетривиальных столбцов элементарных матриц-сомножителей, т. е.,

$$\left[ \begin{array}{cccc} & \overline{U} & & \\ 1 & 2 & -2 & 3 \\ & 1 & 2 & -1 \\ & 1 & -2 & \\ & 1 & & \end{array} \right] = \left[ \begin{array}{ccc} & \overline{U}_4 & & \\ 1 & & 3 & \\ & 1 & -1 & \\ & 1 & -2 & \\ & 1 & & \end{array} \right] \left[ \begin{array}{ccc} & \overline{U}_3 & & \\ 1 & & -2 & \\ & 1 & 2 & \\ & 1 & & \\ & 1 & & \end{array} \right] \left[ \begin{array}{ccc} & \overline{U}_2 & & \\ 1 & 2 & & \\ & 1 & & \\ & 1 & & \\ & 1 & & \end{array} \right],$$

где учтено, что крайний справа сомножитель  $\overline{U}_1$  равен единичной матрице, так как по построению  $\overline{U}$  — верхнетреугольная матрица с единичной диагональю.

Согласно правилу обращения произведения матриц, найдём матрицу  $\bar{U}^{-1}$ :

$$\begin{bmatrix} \bar{U}_2^{-1} \\ 1 & -2 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} \bar{U}_3^{-1} \\ 1 & 2 \\ 1 & -2 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} \bar{U}_4^{-1} \\ 1 & -3 \\ 1 & 1 \\ 1 & 2 \\ 1 & 1 \end{bmatrix}. \quad (3.18)$$

Здесь уже отражён тот факт, что обращение элементарных матриц  $\bar{U}_4$ ,  $\bar{U}_3$  и  $\bar{U}_2$  сводится к простой замене знака на противоположный у каждого нетридиагонального (внедиагонального) элемента.

В действительности же это означает, что сначала – на этапе ① для верхнетреугольной части и на уже рассмотренном этапе ① для нижнетреугольной части (3.17) – результирующий массив из (3.16) пересчитывают по указанным правилам для вычисления  $\bar{L}^{-1}$  и  $\bar{U}^{-1}$ , приводя его к следующему стартовому виду:

$$\begin{bmatrix} 2 & 2 & -2 & 3 \\ 1 & 2 & 2 & -1 \\ 3 & 2 & 3 & -2 \\ 2 & 1 & 2 & 4 \end{bmatrix} \xrightarrow[\textcircled{1}]{\textcircled{1}} \begin{bmatrix} 1/2 & -2 & 2 & -3 \\ -1/2 & 1/2 & -2 & 1 \\ -3/2 & -2/2 & 1/3 & 2 \\ -2/2 & -1/2 & -2/3 & 1/4 \end{bmatrix}. \quad (3.19)$$

Однаковые номера этапов ① и ① здесь подсказывают, что эти подготовительные действия над верхней и нижней частями массива могут быть совмещены по времени в одном цикле.

Процесс поэтапного перемножения в выражении (3.18) отразим в табл. 3.3.

Из табл. 3.3 видно, что после получения верхней треугольной части в (3.19) пересчитывают только следующие элементы: на этапе ② –  $a_{14}$ ,  $a_{24}$  и на этапе ③ –  $a_{13}$ ,  $a_{14}$ . Совмещая операции ② и ② после (3.19), получаем

$$\begin{bmatrix} 1/2 & -2 & 2 & -3 \\ -1/2 & 1/2 & -2 & 1 \\ -3/2 & -2/2 & 1/3 & 2 \\ -2/2 & -1/2 & -2/3 & 1/4 \end{bmatrix} \xrightarrow[\textcircled{2}]{\textcircled{2}} \begin{bmatrix} 1/2 & -2 & 2 & 1 \\ -1/4 & 1/2 & -2 & -3 \\ -1 & -1 & 1/3 & 2 \\ -3/4 & -1/2 & -2/3 & 1/4 \end{bmatrix}.$$

**Совмещение операций ③ и ③**

$$\begin{bmatrix} 1/2 & -2 & 2 & 1 \\ -1/4 & 1/2 & -2 & -3 \\ -1 & -1 & 1/3 & 2 \\ -3/4 & -1/2 & -2/3 & 1/4 \end{bmatrix} \xrightarrow[\textcircled{3}]{\textcircled{3}} \begin{bmatrix} 1/2 & -2 & 6 & 7 \\ -1/4 & 1/2 & -2 & -3 \\ -1/3 & -1/3 & 1/3 & 2 \\ -1/12 & 1/6 & -2/3 & 1/4 \end{bmatrix}$$

Таблица 3.3. Поэтапное перемножение  $\overline{U}_2^{-1} (\overline{U}_3^{-1} \overline{U}_4^{-1})$ 

$$\begin{array}{c}
\overline{U}_2^{-1} \\
\left[ \begin{array}{cc} 1 & -2 \\ & 1 \\ & & 1 \\ & & & 1 \end{array} \right] \times \left[ \begin{array}{ccc} 1 & 2 & \\ & 1 & -2 \\ & & 1 \\ & & & 1 \end{array} \right] \times \left[ \begin{array}{ccc} 1 & & -3 \\ & 1 & 1 \\ & & 1 \\ & & & 1 \end{array} \right]
\end{array}$$
  

$$\begin{array}{c}
\overline{U}_2^{-1} \\
\left[ \begin{array}{cc} 1 & -2 \\ & 1 \\ & & 1 \\ & & & 1 \end{array} \right] \times \left[ \begin{array}{ccc} 1 & 2 & 1 \\ & 1 & -2 & -3 \\ & & 1 & 2 \\ & & & 1 \end{array} \right] \Leftarrow \textcircled{2}
\end{array}$$
  

$$\begin{array}{c}
\overline{U}_2^{-1}(\overline{U}_3^{-1} \overline{U}_4^{-1}) \\
\left[ \begin{array}{cccc} 1 & -2 & 6 & 7 \\ & 1 & -2 & -3 \\ & & 1 & 2 \\ & & & 1 \end{array} \right] \Leftarrow \textcircled{3}
\end{array}$$

завершает операции над верхней треугольной частью, а для нижней треугольной части завершение **④** показано отдельно на стр. 104.

## 3.6 Плохо обусловленные матрицы

Следующие матрицы [12] часто используют для испытания разработанных программ решения систем и обращения матриц в особо сложных условиях, т. е., в таких случаях, когда матрица системы близка к вырожденной матрице.

Обозначения:  $a_{ij}$  — элемент матрицы  $A$ ,  $n$  — её порядок.

**1.** Матрица Гильберта.

$$a_{ij} = 1/(i + j - 1).$$

**2.** Матрица  $A$  с элементами:

$$a_{i,i} = 1 \text{ для } i = 1, 2, \dots, 20;$$

$a_{i,i+1} = 1$  для  $i = 1, 2, \dots, 19$ ;  
 $a_{i,j} = 0$  для остальных значений  $i$  и  $j$ .

$$3. A = \begin{bmatrix} 5 & 4 & 7 & 5 & 6 & 7 & 5 \\ 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 5 & 6 & 7 & 5 & 9 & 10 & 10 \end{bmatrix}.$$

4. Матрица  $A$  с элементами:

$$a_{ii} = 0.01 / [(n - i + 1)(i + 1)];$$

$$a_{ij} = 0 \text{ для } i < j;$$

$$a_{ij} = i(n - j) \text{ для } i > j.$$

5. Матрица из пункта 4, но

$$a_{ij} = j(n - i) \text{ для } i < j.$$

$$6. A = \begin{bmatrix} R & S & T & T \\ S & R & S & T \\ T & S & R & S \\ T & T & S & R \end{bmatrix}, \quad R = \begin{bmatrix} \operatorname{ctg} \theta & \operatorname{cosec} \theta \\ -\operatorname{cosec} \theta & \operatorname{ctg} \theta \end{bmatrix},$$

$$S = \begin{bmatrix} 1 - \operatorname{ctg} \theta & \operatorname{cosec} \theta \\ -\operatorname{cosec} \theta & 1 + \operatorname{ctg} \theta \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Вычисления проводить при  $\theta$  близком к нулю или  $\pi$ .

7. Матрица с параметром  $\alpha$ :

$$a_{ii} = \alpha^{|n-2i|/2};$$

$$a_{1j} = a_{j1} = a_{11}/\alpha^j;$$

$$a_{nj} = a_{jn} = a_{nn}/\alpha^j;$$

$$a_{ij} = 0 \text{ для остальных значений } i \text{ и } j.$$

$$8. a_{ij} = e^{i \cdot j \cdot h}.$$

Вычисления проводить при  $h$  близких к нулю или 1000.

9.  $a_{ij} = c + \log_2(i \cdot j)$ .

Вычисления проводить при больших  $c$ .

$$10. A = \begin{bmatrix} 0.9143 \cdot 10^{-4} & 0 & 0 & 0 \\ 0.8762 & 0.7156 \cdot 10^{-4} & 0 & 0 \\ 0.7943 & 0.8143 & 0.9504 \cdot 10^{-4} & 0 \\ 0.8017 & 0.6123 & 0.7165 & 0.7123 \cdot 10^{-4} \end{bmatrix}.$$

### 3.7 Задание на лабораторный проект № 1

Написать и отладить программу, реализующую ваш вариант метода исключения с выбором главного элемента, для численного решения систем линейных алгебраических уравнений  $Ax = f$ , вычисления  $\det A$  и  $A^{-1}$ . Предусмотреть сообщения, предупреждающие о невозможности решения указанных задач с заданной матрицей  $A$ .

Для выполнения лабораторного проекта необходимо запрограммировать, отладить и протестировать следующие функции (отдельные части программы):

1. Система меню для взаимодействия пользователя с программой и генерация исходных данных задачи.
2. Подпрограмма факторизации (разложения на сомножители) исходной матрицы  $A$ , отвечающая вашему варианту задания.
3. Подпрограмма решения системы линейных алгебраических уравнений  $Ax = f$ .
4. Подпрограмма вычисления определителя матрицы  $\det A$ .
5. Подпрограмма обращения матрицы; способ 1: через решение системы  $AX = I$ .
6. Подпрограмма обращения матрицы; способ 2: через элементарные преобразования разложения.
7. Демонстрационные режимы по выбору пользователя из меню:
  - (a) «Эксперимент 1 – Решение СЛАУ для случайных матриц».
  - (b) «Эксперимент 2 – Решение СЛАУ с плохо обусловленными матрицами».
  - (c) «Эксперимент 3 – Обращение случайных матриц; способ 1».

- (d) «Эксперимент 4 – Обращение случайных матриц; способ 2».
- (e) «Эксперимент 5 – Обращение плохо обусловленных матриц матриц; способ 1».
- (f) «Эксперимент 6 – Обращение плохо обусловленных матриц матриц; способ 2».

8. Сервисные подпрограммы:

- (a) Генерация матриц (с клавиатуры, случайные, плохо обусловленные).
- (b) Сохранение результатов экспериментов в таблицы (в файл) для построения графиков (режим «off-line»).
- (c) Вывод результатов экспериментов в таблицы, но не в файл, а на экран непосредственно по мере вычисления результатов (режим «on-line»).
- (d) Построение графиков.

Уделить особое внимание эффективности программы (в смысле экономии оперативной памяти). Предусмотреть пошаговое выполнение алгоритма исключения с выводом результата на экран.

Выполнить следующие пункты задания.

1. Провести подсчёт фактического числа выполняемых операций умножения и деления при решении системы линейных алгебраических уравнений, сравнить его с оценочным числом ( $n^3/3$ ).

2. Определить скорость решения задач (решение систем линейных алгебраических уравнений, обращение матриц) с учётом времени, затрачиваемого на разложение матрицы. Для этого спроектировать и провести эксперимент, который охватывает матрицы порядка от 5 до 100 (через 5 порядков). Представить результаты в виде таблицы и графика зависимости времени выполнения (в минутах и секундах) от порядка матриц. Таблицу и график вывести на экран.

3. Оценить точность решения систем линейных алгебраических уравнений, имеющих тот же самый порядок, что и задачи из п. 2. Для этого сгенерировать случайные матрицы  $A$ , задать точное решение  $x^*$  и образовать правые части  $f = Ax^*$ . Провести анализ точности вычисленного решения  $x$  от порядка матрицы. Результаты представить в виде таблицы и графика.

Для заполнения матрицы  $A$  использовать случайные числа из диапазона от  $-100$  до  $100$ . В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – порядок матрицы. Для оценки точности использовать норму вектора

$$\|x\|_\infty = \max_i(|x_i|). \quad (3.20)$$

4. Повторить пункт 3 задания для плохо обусловленных матриц (см. подразд. 3.6), имеющих порядок от 4 до 40 с шагом 4.

5. Вычислить матрицу  $A^{-1}$  следующими двумя способами.

Способ 1 – через решение системы  $AX = I$ , где  $I$  – единичная матрица.

Способ 2 – через разложение матрицы  $A$  в произведение элементарных матриц, обращение которых осуществляется отдельными процедурами, а их произведение даёт матрицу  $A^{-1}$ .

Сравнить затраты машинного времени (по числу операций) и точность обращения матриц при использовании указанных способов 1 и 2. Эксперименты провести для случайных матриц порядков от 5 до 100 через 5. Для оценки точности в обоих способах использовать оценочную формулу

$$\|A_{\text{т}}^{-1} - A_{\text{пр}}^{-1}\| \leq \|I - AA_{\text{пр}}^{-1}\| \cdot \|A\|^{-1}. \quad (3.21)$$

Использовать норму матрицы типа «бесконечность», т. е., вычислять ее по следующему выражению:

$$\|A\|_{\infty} = \max_i \left( \sum_{j=1}^n |a_{ij}| \right), \quad (3.22)$$

где  $A_{\text{т}}^{-1}$  – точное значение обратной матрицы, а  $A_{\text{пр}}^{-1}$  – приближенное значение, полученное в результате обращения каждым из способов 1 и 2.

6. Провести подсчёт фактического числа выполняемых операций умножения и деления при обращении матриц первым и вторым способами, сравнить его с оценочным числом ( $n^3$ ).

**Замечание 3.8.** По ходу проведения численных экспериментов на экран дисплея должны выводиться следующие таблицы.

#### Решение систем линейных алгебраических уравнений:

Порядок	Время	Точность	Теоретическое число операций	Реальное число операций
---------	-------	----------	------------------------------	-------------------------

Аналогичная таблица должна быть построена для плохо обусловленных матриц.

#### Обращение матриц:

Порядок	Время		Точность		Число операций		
	способ. 1	способ. 2	способ. 1	способ. 2	способ. 1	способ. 2	теорет.

**Замечание 3.9.** Результаты экспериментов необходимо вывести на экран в форме следующих графиков. Для случая обращения матриц при построении графиков использовать данные из второй таблицы.

### **Графики решения систем линейных алгебраических уравнений:**

- зависимость реального и оценочного числа операций от порядка матрицы (для разных графиков использовать разные цвета);
- зависимость времени решения от порядка матриц;
- зависимость точности решения от порядка матриц. При построении графиков использовать данные из первой таблицы. Для этого их необходимо записать в текстовый файл.

### **Графики для обращения матриц:**

- зависимость реального и оценочного числа операций от порядка матрицы (для разных графиков использовать разные цвета);
- зависимость времени обращения первым и вторым способом от порядка матриц;
- зависимость точности обращения первым и вторым способом от порядка матриц.

## **3.8 Варианты задания на лабораторный проект № 1**

1.  $\bar{LU}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по столбцу.
2.  $\bar{LU}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по строке.
3.  $\bar{LU}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по активной подматрице.
4.  $\bar{LU}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по столбцу.
5.  $\bar{LU}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по строке.
6.  $\bar{LU}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по активной подматрице.

7.  $\overline{LU}$ -разложение на основе гауссова исключения по строкам с выбором главного элемента по строке.
8.  $\overline{LU}$ -разложение по компактной схеме Краута с выбором главного элемента по столбцу.
9.  $\overline{LU}$ -разложение по компактной схеме Краута с выбором главного элемента по строке.
10.  $\overline{LU}$ -разложение по компактной схеме «строка за строкой» с выбором главного элемента по строке.
11.  $\overline{LU}^{-1}$ -разложение  $A = \overline{LU}$  на основе жорданова исключения с выбором главного элемента по столбцу.
12.  $\overline{LU}^{-1}$ -разложение  $A = \overline{LU}$  на основе жорданова исключения с выбором главного элемента по строке.
13.  $\overline{LU}^{-1}$ -разложение  $A = \overline{LU}$  на основе жорданова исключения с выбором главного элемента по активной подматрице.
14.  $\overline{UL}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по столбцу.
15.  $\overline{UL}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по строке.
16.  $\overline{UL}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по активной подматрице.
17.  $\overline{UL}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по столбцу.
18.  $\overline{UL}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по строке.
19.  $\overline{UL}$ -разложение на основе гауссова исключения по столбцам с выбором главного элемента по активной подматрице.
20.  $\overline{UL}$ -разложение на основе гауссова исключения по строкам с выбором главного элемента па строке.
21.  $\overline{UL}$ -разложение по компактной схеме Краута с выбором главного элемента по столбцу.
22.  $\overline{UL}$ -разложение по компактной схеме Краута с выбором главного элемента по строке.
23.  $\overline{UL}$ -разложение по компактной схеме «строка за строкой» с выбором главного элемента по строке.
24.  $\overline{L}^{-1}\overline{U}$ -разложение  $A = \overline{LU}$  на основе жорданова исключения с выбором главного элемента по столбцу.

25.  $\overline{L}^{-1}U$ -разложение  $A = \overline{L}U$  на основе жорданова исключения с выбором главного элемента по строке.

26.  $\overline{L}^{-1}U$ -разложение  $A = \overline{L}U$  на основе жорданова исключения с выбором главного элемента по активной подматрице.

Если нет других указаний преподавателя, выбирайте ваш вариант по вашему номеру в журнале студенческой группы.

### Обозначения:

$L$  – нижняя треугольная матрица,

$U$  – верхняя треугольная матрица,

$\overline{L}$  – нижняя треугольная матрица с единичными элементами на диагонали,

$D$  – диагональная матрица,

$\overline{U}$  – верхняя треугольная матрица с единичными элементами на диагонали.

## 3.9 Методические рекомендации для проекта № 1

В дальнейшем мы используем сокращение СЛАУ: «Система линейных алгебраических уравнений».

Рассмотрим следующие пункты задания последовательно с точки зрения их реализации в проекте.

1. Система меню для взаимодействия пользователя с программой и генерация исходных данных задачи.
2. Функция факторизации матрицы, отвечающая вашему варианту исключения.
3. Функция решения системы линейных алгебраических уравнений.
4. Функция вычисления определителя матрицы.
5. Функция обращения матрицы через решение системы  $AX = I$ .
6. Функция обращения матрицы через элементарные преобразования разложения.

7. Эксперимент 1 «Решение СЛАУ для случайных матриц».
8. Эксперимент 2 «Решение СЛАУ с плохо обусловленными матрицами».
9. Эксперимент 3 «Обращение случайных матриц».

## **Система меню для взаимодействия пользователя с программой**

Возможны различные варианты меню:

1. Меню, в котором взаимодействие с пользователем осуществляется через командную строку.
2. Меню, в котором взаимодействие с пользователем осуществляется через графический интерфейс.

Меню должно включать следующие возможности:

- Ввод с клавиатуры размерности задачи  $n$  ( $n$  – количество уравнений, т. е., число неизвестных в СЛАУ).
- Ввод с клавиатуры элементов квадратной матрицы  $A$  размера  $n \times n$ .
- Ввод с клавиатуры элементов вектора  $b$  размерности  $n \times 1$  - правой части СЛАУ.
- Заполнение матрицы  $A$  и вектора  $b$  случайными числами. Для заполнения матрицы  $A$  использовать случайные числа из диапазона от  $-100$  до  $100$ .

## **Функция факторизации матрицы**

Рассмотрим на примере алгоритма 1 реализацию  $\overline{LU}$ -разложения. Рассмотрим реализацию каждого действия отдельно.

1. Нормируем первую строку матрицы  $A(k - 1)$ :

```
for (int j=k;j<n;j++)
    A[k,j]=A[k,j]/A[k,k];
```

2. Для  $i = k + 1$  до  $n$

Вычитаем первую строку матрицы  $A(k - 1)$ , умноженную на  $a_{ik}^{k-1}$ , из  $i$ -й строки:

```
for (int i=k+1;i<n;i++)
    for (int j=k;j<n;j++)
        A[i,j]=A[i,j]-A[i,k]*A[k,j];
```

Таким образом, программируя отдельные действия с помощью циклов, составляем код функции факторизации матрицы  $A$  по заданному алгоритму.

## Стратегии выбора главного элемента

Стратегия выбора главного элемента по активной подматрице включает в себя стратегию выбора по столбцу и по строке. Поэтому будем рассматривать пример её реализации.

Для максимального элемента нужно запомнить его индексы, например,  $i_{\max}$  и  $j_{\max}$ . После этого элемент матрицы с индексами  $i_{\max}$  и  $j_{\max}$  меняется местами с главным (диагональным) элементом с индексами  $(k, k)$ . Для того чтобы не нарушить связи в матрице линейной системы, необходимо при обмене полностью менять местами строки и столбцы. Однако это делать категорически запрещено. Эффективным способом обмена является введение так называемых векторов перестановок строк и столбцов, которые содержат номера строк и столбцов матрицы  $A$ . Тогда для обмена достаточно поменять всего лишь два элемента в массиве перестановок.

При использовании массива перестановок всюду в программе обращаемся к элементам матрицы следующим образом: для первой стратегии  $A(p(i), j)$ , для второй стратегии  $A(i, q(j))$ , а для третьей стратегии  $A(p(i), q(j))$ , где  $p$  и  $q$  – векторы перестановок строк и столбцов, соответственно. Рассмотрим реализацию третьей стратегии как более общей:

### Листинг 3.1.

```
// Массивы перестановок: p - массив номеров строк,
// q - массив номеров столбцов
int [] p = new int[n];
int [] q = new int[n];
    //вспомогательные переменные
int imax = 0;
int jmax = 0;
int buf = 0;
// Заполняем массивы перестановок начальными значениями
// перед началом факторизации
```

```

for (int i=0;i<n;i++) {
    p[i]=i;
    q[i]=i;
}
// Этот фрагмент вставляем в алгоритм факторизации
// n - размер задачи, k - номер текущего шага алгоритма
imax=k;
jmax=k;

for (int i=k;i<n;i++)
    for (int j=k;j<n;j++)
        if(Math.Abs(A[p[i],q[j]])> Math.Abs(A[p[imax],q[jmax]])) {
            imax=i;
            jmax=j;
        }
// Обмен
if (imax!=k) {
    int buf=p[k];
    p[k]=p[imax];
    p[imax]=buf;
}
if (jmax!=k) {
    int buf=q[k];
    q[k]=q[jmax];
    q[jmax]=buf;
}

```



## Решение системы линейных алгебраических уравнений

Рассмотрим детали реализации алгоритма нахождения решения СЛАУ по  $LU$ -разложению. Сначала рассмотрим нахождение решения без учёта выбора главного элемента. Решение будем искать в два этапа: сначала находим решение для системы с нижней треугольной матрицей, затем находим решение с верхней треугольной матрицей. Приведём математическое обоснование:

$$Ax = b \Rightarrow \overline{LU}x = b \Rightarrow y \triangleq \overline{U}x \Rightarrow Ly = b \Rightarrow y \Rightarrow \overline{U}x = y \Rightarrow x$$

**Первый проход:** известны матрица  $L$  (нижняя треугольная часть факторизованной матрицы  $A$ ) и вектор  $b$ , находим неизвестный вектор  $y$  с помощью прямой подстановки:

### Листинг 3.2.

```
// метод прямой подстановки (скалярных произведений)
for (int i=0;i<n;i++) {
    float sum=0;
    for (int j=0;j<=i;j++)
        sum=sum+A[i,j]*y[j];
    y[i]=(b[i]-sum)/A[i,i];
}
```



**Второй проход:** известны матрица  $\bar{U}$  (верхняя треугольная часть факторизованной матрицы  $A$ ) и вектор  $y$ , находим неизвестный вектор  $x$  с помощью обратной подстановки:

### Листинг 3.3.

```
// метод обратной подстановки (скалярных произведений)
for (int i=n-1;i>-1;i--) {
    float sum=0;
    for (int j=i+1;j<n;j++)
        sum=sum+A[i,j]*x[j];
    x[i]=y[i]-sum; // Найденное решение находится в массиве x
}
```



Можно использовать и другие способы программной реализации методов прямой и обратной подстановки (см. [6], с. 67–69). Аналогично реализуются алгоритмы нахождения решения и для других вариантов факторизации.

Теперь будем учитывать, что при факторизации была использована стратегия выбора главного элемента по активной подматрице. В этом случае при перестановке строк в СЛАУ необходимо переставлять соответствующие элементы в векторе  $b$  (чтобы не потерять связи между неизвестными и уравнениями в линейной системе). В программе нужно обращаться к массиву  $b$  также через вектор перестановок строк, т. е.,  $b(p(i))$ . Если в СЛАУ переставляют столбцы

матрицы коэффициентов, то таким же образом должны быть переставлены и элементы вектора  $x$ . Тогда один из вариантов программной реализации алгоритма нахождения СЛАУ при выборе главного элемента по активной подматрице можно записать так:

**Первый проход:** известны матрица  $L$  и вектор  $b$ , находим неизвестный вектор  $y$  с помощью прямой подстановки, учитываем перестановки в элементах матрицы и вектора:

### Листинг 3.4.

```
// метод прямой подстановки (скалярных произведений)
for (int i=0;i<n;i++) {
    float sum=0;
    for (int j=0;j<=i;j++)
        sum=sum+A[p[i],q[j]]*y[j];
    y[i]=(b[p[i]]-sum)/A[p[i],q[i]];
}
```



**Второй проход:** известны матрица  $\bar{U}$  и вектор  $y$ , находим неизвестный вектор  $x$  с помощью обратной подстановки, учитываем перестановки в элементах матрицы и вектора:

### Листинг 3.5.

```
// метод обратной подстановки (скалярных произведений)
for (int i=n-1;i>-1;i--) {
    float sum=0;
    for (int j=i+1;j<n;j++)
        sum=sum+A[p[i],q[j]]*x[q[j]];
    x[q[i]]=y[i]-sum;
}
// Найденное решение находится в массиве x
```



## Определитель матрицы

Если разложение  $A = L\bar{U}$  уже известно, определитель находится просто как произведение диагональных элементов в факторизованной матрице, так как

$$\det A = \det L \cdot \det \bar{U} = \prod_{i=1}^n l_{ii} \cdot 1 = \prod_{i=1}^n l_{ii} = \prod_{i=1}^n a_{ii}.$$

Последнее равенство записано с учётом того, что исходная матрица  $A$  замещена элементами результирующих матриц  $L$  и  $\bar{U}$  и что определитель верхней треугольной матрицы с единичной диагональю равен единице, а определитель нижней треугольной матрицы равен произведению её диагональных элементов.

Если мы используем любую из трёх стратегий выбора главного элемента, то при работе алгоритма факторизации происходит обмен строк и/или столбцов. При этом каждый раз при обмене знак определителя меняется на противоположный. При написании программы можно объявить глобальный флаг `znak`, в котором будет сохраняться значение  $-1$  или  $+1$ , в зависимости от того, чётное или нечётное количество перестановок было сделано на данный момент. Этот флаг следует изменять в той части программы, где происходит обмен значений в векторах перестановок. Затем при вычислении определителя произведение диагональных элементов нужно умножить на этот флаг.

### Листинг 3.6.

```
// перед началом алгоритма факторизации
int znak=1;
// при обмене в векторах перестановок
if (imax!=k) {
    znak=-znak;
    int buf=p[k];
    p[k]=p[imax];
    p[imax]=buf;
}
if (jmax!=k) {
    znak=-znak;
    int buf=q[k];
    q[k]=q[jmax];
    q[jmax]=buf;
}
// теперь вычисляем определитель
```

```

float det=1;
for (int i=0;i<n;i++)
    det=det*A[p[i],q[i]];
det=det*znak;

```



## Обращение матрицы через решение системы

Предположим, что мы уже нашли разложение  $A = L\bar{U}$  и в данный момент в массиве  $A$  находится факторизованная матрица  $A$ . Более того, у нас уже реализован алгоритм нахождения решения СЛАУ. Будем искать обратную матрицу из следующих соображений. Обозначим обратную матрицу через  $X$ . Тогда верно тождество  $AX = I$ , где  $I$  – единичная матрица. Разобьём матрицы  $X$  и  $I$  на столбцы и запишем систему, состоящую из  $n$  СЛАУ:

$$\begin{cases} Ax_1 = e_1 \\ Ax_2 = e_2 \\ \dots \\ Ax_n = e_n \end{cases}$$

где  $e_k$  –  $k$ -й столбец единичной матрицы  $I$ ,  $x_k$  –  $k$ -й столбец искомой обратной матрицы  $A^{-1}$ .

Каждую из этих  $n$  СЛАУ решаем уже имеющимся алгоритмом, т. е., поочереди находим столбцы обратной матрицы и собираем их вместе. Так мы находим обратную матрицу (это первый способ).

Для программной реализации необходим дополнительный массив  $A1$ , в который будем записывать найденные столбцы обратной матрицы. Далее в цикле заполняем массив  $b$  элементами очередного столбца единичной матрицы, решаем СЛАУ и получаем в массиве  $x$  очередной столбец обратной матрицы, затем записываем его в матрицу  $A1$ .

## Обращение матрицы через элементарные преобразования

Предположим, что мы уже нашли разложение, и в данный момент в массиве  $A$  находится преобразованная матрица  $A$ . Необходимо на месте разложения в массиве  $A$  вычислить обратную матрицу  $A1$ . Пример вычисления обратной матрицы данным способом подробно рассмотрен в подразд. 3.5 на с. 101. Смысл

алгоритма заключается в следующем (пока не учитываем стратегию выбора главного элемента):

Пусть  $A = LU$ . Тогда  $A^{-1} = \bar{U}^{-1}L^{-1}$ . Алгоритм вычисления обратной матрицы можно разделить на четыре этапа:

1. Выполняем подготовку элементов матрицы для вычисления обратной. Для этого вычисляем обратные к элементарным матрицам следующим образом: в матрице  $\bar{U}$  меняем знаки на противоположные у всех наддиагональных элементов; в матрице  $L$  каждый диагональный элемент заменяется на обратный по величине, затем полученное число берём с противоположным знаком и умножаем на каждый поддиагональный элемент.
2. Вычисляем матрицу, обратную к  $\bar{U}$ , в верхней треугольной части массива  $A$  (так как у данной матрицы единичная диагональ, то у обратной матрицы также будет единичная диагональ, и её можно не хранить в массиве).

### Листинг 3.7.

```
for (int k=n;k>-1;k-=2)
    for (int i=0;i>=k-2;i++)
        for (int j=k;j<n;j++)
            A[i,j]=A[i,j]+A[i,k-1]*A[k-1,j] ;
```



3. Вычисляем матрицу, обратную к  $L$ , в нижней треугольной части (вместе с диагональю) массива  $A$ .

### Листинг 3.8.

```
for (int k=1;k<n-1;k++) {
    for (int i=k+2;i<n;i++) {
        for (int j=1;j<=k;j++) {
            A[i,j]=A[i,j]+A[i,k+1]*A[k+1,j] ;
        }
    }
    for (int j=1;j<=k;j++)
        A[k+1,j]=A[k+1,j]*A[k+1,k+1] ;
}
```



4. Перемножаем в одном массиве матрицы  $\overline{U}^{-1}$  и  $L^{-1}$ . Получаем искомую матрицу  $A_1$ . При вычислении нужно правильно задать границы изменения индексов для верхней треугольной и нижней треугольной частей массивов.

### Листинг 3.9.

```

for (int i=0;i<n;i++) {
    for (int j=0;j<n;j++) {
        if(i<j) {
            float sum=0;
            for (int k=j;k<n;k++)
                sum=sum+A[i,k]*A[k,j];
        }
        if(i>=j) {
            sum=A[i,j];
            for (int k=i+1;k<n;k++)
                sum=sum+A[i,k]*A[k,j];
        }
        A[i,j]=sum;
    }
}

```



Для того чтобы учесть стратегию выбора главного элемента, необходимо в программном коде обращаться к элементам матрицы  $A$  как  $A(p(i), q(j))$ . Так как при факторизации исходной матрицы  $A$  использовались перестановки строк и столбцов, в результате вычислений мы получим матрицу, обратную к матрице  $PAQ$ , где  $P$  и  $Q$  – матрицы перестановок строк и столбцов. Следовательно,

$$PAQ = L\overline{U} \Rightarrow \overline{U}^{-1}L^{-1} = Q^{-1}A^{-1}L^{-1} \Rightarrow A^{-1} = QU^{-1}L^{-1}P.$$

Тогда  $(i, j)$ -му элементу обратной матрицы  $A^{-1}$  соответствует элемент массива  $A(p(q_1(i)), q(p_1(j)))$ , где  $p_1, q_1$  – векторы обратных перестановок, элементы которых вычисляются как  $p_1(p(i))=i$  и  $q_1(q(j))=j$ .

Программная реализация остальных вариантов разложения может быть построена с помощью модификации приведённого программного кода (листинг 3.9).

## Эксперимент 1 «Решение СЛАУ для случайных матриц»

Требуется написать функцию для проведения первого эксперимента с целью исследования скорости и погрешности решения СЛАУ. Подробное описание эксперимента можно найти в разд. 3.7 на с. 109. Перед реализацией эксперимента необходимо написать процедуру, вычисляющую погрешность решения СЛАУ.

### Методика проведения эксперимента:

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

#### Решение СЛАУ для случайных матриц:

Порядок	Время	Точность	Теоретическое число операций	Реальное число операций
---------	-------	----------	---------------------------------	----------------------------

Каждая строка в таблице будет зависеть от  $n$  – количества уравнений в системе. Число  $n$  должно изменяться от 5 до 100 через 5 порядков, т. е., в таблице будет 20 строк.

2. Присвоить  $n$  очередное значение.
3. Заполнить матрицу  $A$  случайными числами в соответствии с п. 1 из перечня заданий на с. 114.
4. Выполнить факторизацию матрицы  $A$  в соответствии с п. 2 из перечня заданий на с. 114.
5. Найти решение СЛАУ  $x$  в соответствии с п. 3 из перечня заданий на с. 114.
6. Подсчитать теоретическое число операций умножения и деления по формуле  $n^3/3$ .
7. Подсчитать реальное число операций умножения и деления с помощью специальных счётчиков, которые вы добавите в функции факторизации и решения.
8. Подсчитать скорость решения задачи как сумму времени, затраченного на разложение матрицы, и времени решения СЛАУ.
9. Оценить погрешность решения СЛАУ:

- Задать точное решение  $x^*$ . В качестве точного решения нужно взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – размер очередной матрицы.
  - Образовать правые части  $b := Ax^*$ , где матрица  $A$  известна из п. 3.
  - Найти решение СЛАУ  $Ax = b$ .
  - Вычислить погрешность решения СЛАУ как максимальный модуль разности компонент вектора точного решения и вектора найденного решения.
10. Добавить в таблицу и файл, предназначенный для хранения экспериментальных данных, полученные экспериментальные значения: порядок  $n$ , время выполнения, погрешность решения СЛАУ, теоретическое и реальное число операций.
11. Повторить пункты 2–8 для всех значений  $n$ .
12. Построить следующие графики решения СЛАУ:
- зависимость реального и оценочного числа операций от размера матрицы (для разных графиков использовать разные цвета);
  - зависимость времени решения от размера матриц;
  - зависимость погрешности решения от размера матриц.
- При построении графиков использовать данные из файла с экспериментальными данными.
13. Проанализировать полученные данные и сделать содержательные выводы об эффективности и качестве реализованных численных методов.

## Эксперимент 2 «СЛАУ с плохо обусловленными матрицами»

Написать реализацию второго эксперимента для исследования скорости и погрешности решения СЛАУ в случае, когда матрица  $A$  является плохо обусловленной. Подробное описание эксперимента можно найти в разд. 3.7 на с. 109. Перед проведением эксперимента написать функцию, вычисляющую погрешность решения СЛАУ (см. эксперимент 1, с. 124).

### Методика проведения эксперимента:

Для каждой из 10 плохо обусловленных матриц (см. подразд. 3.6 с. 107) выполнить следующее:

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

**Решение СЛАУ с плохо обусловленными матрицами:**

Порядок	Время	Точность	Теоретическое число операций	Реальное число операций
---------	-------	----------	---------------------------------	----------------------------

Каждая строка в таблице будет зависеть от  $n$  – количества уравнений в системе. Если матрица  $A$  имеет фиксированный размер (матрицы с номерами 2, 3, 6, 10), тогда  $n$  равно размеру матрицы, и в таблице будет только одна строка. Если порядок матрицы не фиксирован (матрицы с номерами 1, 4, 5, 7, 8, 9), тогда число  $n$  должно изменяться от 4 до 40 через 4 порядка, т. е., в таблице будет 10 строк.

2. Присвоить  $n$  очередное значение.
3. Заполнить матрицу  $A$  в соответствии с её номером и п. 1 из перечня заданий на с. 114 (для заданной матрицы!).
4. Выполнить факторизацию матрицы  $A$  в соответствии с п. 2 из перечня заданий на с. 114.
5. Найти решение СЛАУ  $x$  в соответствии с п. 3 из перечня заданий на с. 114.
6. Подсчитать теоретическое число операций умножения и деления по формуле  $n^3/3$ .
7. Подсчитать реальное число операций умножения и деления с помощью специальных счётчиков, которые вы добавите в функции факторизации и решения.
8. Подсчитать скорость решения задачи как сумму времени, затраченного на разложение матрицы, и времени решения СЛАУ (см. эксперимент 1, с. 124).
9. Оценить погрешность решения СЛАУ (см. эксперимент 1, с. 124):
  - Задать точное решение  $x^*$ . В качестве точного решения нужно взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – размер очередной матрицы.
  - Образовать правые части  $b := Ax^*$ , где матрица  $A$  известна из п. 3.
  - Найти решение СЛАУ  $Ax = b$ .

- Вычислить погрешность решения СЛАУ как максимальный модуль разности компонент вектора точного решения и вектора найденного решения.
10. Добавить в таблицу и файл, предназначенный для хранения экспериментальных данных, полученные экспериментальные значения: порядок  $n$ , время выполнения, погрешность решения СЛАУ, теоретическое и реальное число операций.
11. Повторить пункты 2-8 для всех значений  $n$ .
12. Построить следующие графики решения СЛАУ:
- зависимость реального и оценочного числа операций от размера матрицы (для разных графиков использовать разные цвета);
  - зависимость времени решения от размера матриц;
  - зависимость погрешности решения от размера матриц.
- При построении графиков использовать данные из файла с экспериментальными данными.
13. Проанализировать полученные данные и сделать содержательные выводы об эффективности и качестве реализованных численных методов.

### **Эксперимент 3 «Обращение случайных матриц»**

Реализовать третий эксперимент для исследования скорости и погрешности алгоритмов обращения матриц. Подробное описание эксперимента можно найти в разд. 3.7 на с. 109.

Перед реализацией эксперимента написать процедуру, вычисляющую погрешность найденной обратной матрицы. Формулы (3.21), (3.22) в разд. 3.7 позволяют оценить сверху погрешность обращения матрицы  $A$ . Для этого необходимо, в соответствии с этими формулами, сначала умножить исходную матрицу на найденную обратную и вычесть результат из единичной матрицы, затем найти норму найденной разности и поделить её на норму исходной матрицы. Норма матрицы вычисляется по формуле (3.22) на с. 111 – это максимальная сумма модулей элементов строк.

#### **Методика проведения эксперимента:**

Для каждой из случайных матриц выполнить следующие пункты задания:

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

**Обращение матриц:**

Порядок	Время		Точность		Число операций		
	спос. 1	спос. 2	спос. 1	спос. 2	спос. 1	спос. 2	теорет.

Каждая строка в таблице будет зависеть от  $n$  – размера матрицы. Число  $n$  должно изменяться от 5 до 100 через 5 порядков, т. е., в таблице будет 20 строк.

2. Присвоить  $n$  очередное значение.
3. Заполнить матрицу  $A$  случайными числами в соответствии с п. 1 из перечня заданий на с. 114.
4. Сохранив предварительно копию исходной матрицы  $A$ , выполнить её факторизацию в соответствии с п. 2 из перечня заданий на с. 114.
5. Найти обратную матрицу первым способом в соответствии с п. 5 из перечня заданий на с. 114.
6. Подсчитать скорость обращения матрицы (см. эксперимент 1, с. 124) первым способом.
7. Найти погрешность вычисления обратной матрицы.
8. Найти обратную матрицу вторым способом в соответствии с п. 6 из перечня заданий на с. 114.
9. Подсчитать скорость обращения матрицы (см. эксперимент 1, с. 124) вторым способом.
10. Найти погрешность вычисления обратной матрицы.
11. Подсчитать теоретическое число операций умножения и деления по формуле  $n^3$ .
12. Подсчитать реальное число операций умножения и деления с помощью специальных счётчиков, которые вы добавите в функции факторизации и обращения первым и вторым способами.

13. Добавить в таблицу полученные экспериментальные данные: порядок  $n$ , время и погрешность обращения первым и вторым способами, теоретическое и реальное число операций при обращении первым и вторым способами.
14. Пункты 2-8 повторить для всех значений  $n$ .
15. Построить следующие графики для режима обращения матриц::
- зависимость реального и оценочного числа операций от размера матрицы (для разных графиков использовать разные цвета);
  - зависимость времени обращения первым и вторым способами от размера матрицы;
  - зависимость погрешности обращения первым и вторым способами от размера матрицы (см. эксперимент 1, с. 124).

При построении графиков использовать данные из файла с экспериментальными данными.

16. Проанализировать полученные данные и сделать содержательные выводы об эффективности и качестве реализованных численных методов.

## 3.10 Тестовые задачи для проекта № 1

Используйте приводимые ниже задачи в двух режимах:

- для контроля собственного понимания алгоритма,
- для контроля правильности вашего программирования.

### Задача 1

*Для матрицы*

$$A = \begin{pmatrix} 2 & 0 & 2 \\ 4 & -1 & 3 \\ -2 & -3 & -2 \end{pmatrix}$$

*выполнить следующее:*

- a. Построить  $\overline{L}U$ -разложение матрицы  $A$  ( $\overline{L}$  с единицами на диагонали).

б. С помощью  $\overline{L}U$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, 0, -3)^T$ .

в. С помощью  $\overline{L}U$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 2

Для матрицы

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 6 & 2 & 1 \\ -2 & -2 & -1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $\overline{L}U$ -разложение матрицы  $A$  ( $\overline{L}$  с единицами на диагонали).

б. С помощью  $\overline{L}U$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, 3, 1)^T$ .

в. С помощью  $\overline{L}U$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 3

Для матрицы

$$A = \begin{pmatrix} -1 & 4 & 1 \\ 1 & -2 & 2 \\ -2 & 8 & 3 \end{pmatrix}$$

выполнить следующее:

а. Построить  $\overline{L}U$ -разложение матрицы  $A$  ( $\overline{L}$  с единицами на диагонали).

б. С помощью  $\overline{L}U$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-2, -1, -5)^T$ .

- в. С помощью  $\overline{LU}$ -разложения найти матрицу  $A^{-1}$  и вычислить число обусловленности матрицы  $A$  ( $M_A$ ) в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 4

Для матрицы

$$A = \begin{pmatrix} 3 & 6 & 2 \\ -5 & -10 & -4 \\ 1 & 3 & 1 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $\overline{U}L$ -разложение матрицы  $A$  ( $\overline{U}$  с единицами на диагонали).
- б. С помощью  $\overline{U}L$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (10, -16, 5)^T$ .

- в. С помощью  $\overline{U}L$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 5

Для матрицы

$$A = \begin{pmatrix} 3 & 0 & 3 \\ -1 & 1 & -2 \\ 1 & 2 & 1 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $L\overline{U}$ -разложение матрицы  $A$  ( $\overline{U}$  с единицами на диагонали).
- б. С помощью  $L\overline{U}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, -2, -2)^T$ .

- в. С помощью  $L\overline{U}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 6**

Для матрицы

$$A = \begin{pmatrix} 3 & 0 & 9 \\ -1 & 1 & -5 \\ 1 & 2 & 0 \end{pmatrix}$$

выполнить следующее:

- a. Построить  $L\bar{U}$ -разложение матрицы  $A$  ( $\bar{U}$  с единицами на диагонали).
- б. С помощью  $L\bar{U}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (12, -7, -1)^T$ .

- в. С помощью  $L\bar{U}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 7**

Для матрицы

$$A = \begin{pmatrix} -3 & 1 & 1 \\ 2 & 1 & 2 \\ 4 & 0 & 2 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $U\bar{L}$ -разложение матрицы  $A$  ( $\bar{L}$  с единицами на диагонали).
- б. С помощью  $U\bar{L}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (5, 2, 0)^T$ .

- в. С помощью  $U\bar{L}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 8**

Для матрицы

$$A = \begin{pmatrix} 2 & 2 & -4 \\ 1 & 2 & -2 \\ 2 & 1 & -1 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $\bar{L}\bar{U}^{-1}$ -разложение матрицы  $A$  ( $\bar{U}^{-1}$  с единицами на диагонали).

- б. С помощью  $\bar{L}\bar{U}^{-1}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, -1, -2)^T$ .

- в. С помощью  $\bar{L}\bar{U}^{-1}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 9**

Для матрицы

$$A = \begin{pmatrix} 6 & 1 & -1 \\ 5 & 1 & -2 \\ -8 & 0 & 4 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $\bar{L}^{-1}\bar{U}$ -разложение матрицы  $A$  ( $\bar{L}^{-1}$  с единицами на диагонали).

- б. С помощью  $\bar{L}^{-1}\bar{U}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-3, 0, 0)^T$ .

- в. С помощью  $\bar{L}^{-1}\bar{U}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### 3.11 Заключение по разделу 3

В данном разделе рассмотрены стандартные алгоритмы  $LU$ -разложения матрицы, численно эквивалентные методу Гаусса последовательного исключения неизвестных в системе линейных алгебраических уравнений. Даны теоретические сведения, достаточные для выполнения лабораторного проекта № 1.

В проекте № 1 предлагается выполнить программирование  $LU$ -разложения и на этой основе решить классические задачи вычислительной линейной алгебры: найти решение СЛАУ, найти обратную матрицу и вычислить её определитель. При этом матрицу системы предлагается формировать тремя различными способами: вводить «вручную» с клавиатуры компьютера, генерировать случайным образом или же из списка специальных – плохо обусловленных – матриц.

Этот проект является базовым для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

# 4

## Проект № 2 «Разложения Холесского»

### 4.1 Положительно определённые матрицы

**Определение 4.1.** Симметрическая матрица  $P$ ,  $P(n, n)$ <sup>1</sup>, называется положительно определённой (ПО), если и только если  $x^T Px > 0$  для всех  $n$ -векторов  $x$  с  $\|x\| > 0$ .

В определении 4.1 выражение  $x^T Px$  есть квадратичная форма в терминах переменных элементов некоторого (произвольного) вектора  $x \in \mathbb{R}^n$ . Например, для размерности  $n = 3$  этого вектора имеем

$$\begin{aligned} x^T Px &= \sum_{i,j=1}^n p(i, j)x_i x_j = \\ &= p(1, 1)x_1^2 + 2p(1, 2)x_1 x_2 + 2p(1, 3)x_1 x_3 + \\ &\quad + p(2, 2)x_2^2 + 2p(2, 3)x_2 x_3 + \\ &\quad + p(3, 3)x_3^2. \end{aligned}$$

Этот очевидный способ раскрытия формулы для  $x^T Px$  справелив и в общем случае (для любого целого  $n \geq 2$ ), – он обобщает формулу квадрата суммы двух или более чисел.

#### Свойства ПО матриц

Когда  $P$  есть симметрическая матрица, обозначение  $P > 0$  означает, что  $P$  является положительно определённой.

**Свойство А.**  $P > 0$  тогда и только тогда, когда все собственные числа матрицы  $P$  положительны.

---

<sup>1</sup> Т. е.,  $P$  имеет размер  $(n \times n)$ .

**Свойство В.** Если  $P > 0$ , то все диагональные элементы матрицы  $P$  положительны.

**Свойство С.** Если матрица  $M$  невырожденная и  $P > 0$ , то  $M^T P M > 0$ .

**Свойство D.** Если  $P > 0$ , то  $P^{-1}$  существует и  $P^{-1} > 0$ .

**Свойство Е.** Если  $P > 0$ , то матрица, полученная вычеркиванием  $i$ -й строки и  $i$ -го столбца, также является положительно определённой.

**Свойство F.** Если  $P > 0$  и  $\rho(i, j) = p(i, j)/[p(i, i)p(j, j)]^{1/2}$  с индексами элементов  $i, j = 1, \dots, n$  при  $i \neq j$ , то  $|\rho(i, j)| < 1$ , при этом  $\rho(i, j)$  называются коэффициентами корреляции.

### Признаки положительной определённости матриц

**Критерий Сильвестра.** Чтобы матрица  $P$  была положительно определённой, необходимо и достаточно, чтобы все её главные миноры были положительны.

**Достаточное условие.** Диагональное преобладание, т. е., свойство

$$\forall i = 1, \dots, n : \quad p(i, i) > \sum_{j=1, j \neq i}^n |p(i, j)|,$$

влечёт положительную определённость матрицы  $P = [p(i, j)]$ .

## 4.2 Квадратные корни из $P$ и алгоритмы Холесского

**Определение 4.2.** Если матрица  $P$  может быть представлена как

$$P = S S^T \tag{4.1}$$

с квадратной матрицей  $S$ , то  $S$  называют квадратным корнем из  $P$  [19]. Квадратные корни матриц, когда они существуют, определяются равенством (4.1) не единственным образом, так как, если  $S$  удовлетворяет равенству (4.1) и  $T$  есть любая ортогональная ( $TT^T = I$  и  $T^T T = I$ ) матрица, то  $ST$  также удовлетворяет (4.1).

**Замечание 4.1.** Хотя определение 4.2, т. е., формула (4.1), часто используется, оно не является универсальным. Обобщения заключаются в отказе от квадратной формы матрицы  $S$  или (если она остается квадратной) в допущении комплексных значений её элементов. В последнем случае пишут  $P = SS^H$ , где  $S^H$  обозначает комплексно сопряжённую при транспонировании матрицу. Ограничения в определении квадратного корня из матрицы могут заключаться в различных требованиях к её форме. Например, можно требовать симметричности  $S = S^T$  или эрмитовости  $S = S^H$ . В случае симметричности, т. е., при  $S = S^T$ , квадратный корень из матрицы  $P$  обозначают  $S^{1/2}$ , т. е., пишут  $P = SS^{1/2}$ . Если же требовать, чтобы  $S$  в (4.1) имела треугольную форму, то в (4.1) приходим к четырём вариантам разложения Холесского [13, 16].

**Определение 4.3.** Разложением Холесского принято называть любое из следующих представлений положительно определённой матрицы  $P$ :

$$P = LL^T, \quad P = \overline{L} D \overline{L}^T, \quad P = UU^T, \quad P = \overline{U} D \overline{U}^T, \quad (4.2)$$

где  $L$  – нижняя треугольная матрица с положительными элементами на диагонали,  $U$  – верхняя треугольная матрица с положительными элементами на диагонали,  $\overline{L}$  – нижняя треугольная матрица с единичными элементами на диагонали,  $D$  – диагональная матрица с положительными элементами на диагонали,  $\overline{U}$  – верхняя треугольная матрица с единичными элементами на диагонали.

**Теорема 4.1 (Нижнее треугольное разложение Холесского).** Симметрическая матрица  $P > 0$  имеет разложение  $P = LL^T$ , где  $L$  – нижняя треугольная матрица, при этом такое разложение на сомножители с положительными диагональными элементами в  $L$  даётся следующим алгоритмом.

Для  $j = 1, \dots, n-1$  рекуррентно выполнять цикл, образованный следующим упорядоченным набором выражений:

$$\left. \begin{aligned} L(j, j) &= P(j, j)^{1/2}, \\ L(k, j) &= P(k, j)/L(j, j), \quad k = j + 1, \dots, n, \\ P(i, k) &:= P(i, k) - L(i, j)L(k, j) \end{aligned} \right\} \begin{aligned} k &= j + 1, \dots, n \\ i &= k, \dots, n \end{aligned} \quad L(n, n) = P(n, n)^{1/2}.$$

**Замечание 4.2.** Приведённая формулировка алгоритма использует обозначения элементов матриц  $P$  и  $L$  для наглядности. Это не должно создавать впечатления, что данные матрицы хранятся в памяти по отдельности; наоборот,

в лабораторном проекте все приведённые вычисления должны проводиться в одном и том же массиве. Это значит, что элементы матрицы  $P$  должны замещаться элементами матрицы  $L$  по мере вычисления последних.

**Замечание 4.3.** Легко видеть, что если  $L_1$  и  $L_1$  суть два различных варианта факторизации матрицы  $P > 0$ , то

$$L_1 = L_2 \operatorname{diag} [\pm 1, \dots, \pm 1],$$

т. е., в приведённом алгоритме можно брать  $L(j, j) = \pm P(j, j)^{1/2}$ . Обычно рекомендуют выбирать единственное решение, отвечающее положительным диагональным элементам матрицы  $L$ .

Внимательное прочтение теоремы 4.1 обнаруживает, что данный алгоритм требует вычисления  $n$  квадратных корней, что замедляет вычисления. Этот недостаток устраняется, если перейти ко второму варианту разложения Холесского из общего списка (4.2).

**Следствие 4.1** (*Нижнее треугольное разложение Холесского без операции квадратного корня*). Симметрическая матрица  $P > 0$  имеет разложение  $P = \overline{L} D \overline{L}^T$ , где  $\overline{L}$  – нижняя треугольная матрица с единичной диагональю и  $D = \operatorname{diag} (d_1, \dots, d_n)$ , при этом элементы матриц  $\overline{L}$  и  $D$  даются следующим алгоритмом.

Для  $j = 1, \dots, n-1$  рекуррентно выполнять цикл, образованный следующим упорядоченным набором выражений:

$$\left. \begin{aligned} d_j &= P(j, j), & \overline{L}(j, j) &= 1, \\ P(i, k) &:= P(i, k) - \overline{L}(i, j)P(k, j) & \left\{ \begin{array}{l} k = j+1, \dots, n \\ i = k, \dots, n \end{array} \right\} & \begin{cases} d(n) = P(n, n), \\ \overline{L}(n, n) = 1. \end{cases} \\ \overline{L}(k, j) &= P(k, j)/d(j), & k &= j+1, \dots, n \end{aligned} \right\}$$

Данный результат легко получается из теоремы 4.1 с использованием обозначений  $d_j = L(j, j)^2$  и  $\overline{L}(i, j) = L(i, j)/L(j, j)$ .

Следующий аналог теоремы 4.1 формулирует алгоритм третьей версии разложения Холесского из списка (4.2).

**Теорема 4.2** (*Верхнее треугольное разложение Холесского*). Симметрическая матрица  $P > 0$  имеет разложение  $P = UU^T$ , где  $U$  – верхняя треугольная матрица, при этом такое разложение на сомножители с положительными диагональными элементами в  $U$  даётся следующим алгоритмом.

Для  $j = n, n - 1, \dots, 2$  рекуррентно выполнять цикл, образованный следующим упорядоченным набором выражений:

$$\left. \begin{array}{l} U(j, j) = P(j, j)^{1/2}, \\ U(k, j) = P(k, j)/U(j, j), \quad k = 1, \dots, j - 1, \\ P(i, k) := P(i, k) - U(i, j)U(k, j) \end{array} \right\} \begin{array}{l} k = 1, \dots, j - 1 \\ i = 1, \dots, k \end{array} \quad U(1, 1) = P(1, 1)^{1/2}.$$

Аналогично следствию 4.1, зафиксируем последний из четырёх вариантов разложения Холесского (4.2).

**Следствие 4.2** (*Верхнее треугольное разложение Холесского без операции квадратного корня*). Симметрическая матрица  $P > 0$  имеет разложение  $P = \overline{U} D \overline{U}^T$ , где  $\overline{U}$  – верхняя треугольная матрица с единичной диагональю и  $D = \text{diag}(d_1, \dots, d_n)$ , при этом элементы матриц  $\overline{U}$  и  $D$  даются следующим алгоритмом.

Для  $j = n, n - 1, \dots, 2$  рекуррентно выполнять цикл, образованный следующим упорядоченным набором выражений:

$$\left. \begin{array}{l} d_j = P(j, j), \quad \overline{U}(j, j) = 1, \\ \overline{U}(k, j) = P(k, j)/d(j), \quad k = 1, \dots, j - 1, \\ P(i, k) := P(i, k) - \overline{U}(i, j)\overline{U}(k, j)d_j \end{array} \right\} \begin{array}{l} k = 1, \dots, j - 1 \\ i = 1, \dots, k \end{array} \quad \begin{array}{l} d(1) = P(1, 1), \\ \overline{U}(1, 1) = 1. \end{array}$$

### 4.3 Программная реализация алгоритмов Холесского

В справочных целях включаем реализацию трёх из четырёх приведённых выше алгоритмов на языке FORTRAN [16]. Эти примеры реализации могут помочь студентам написать свои собственные программы на других языках высокого уровня при выполнении лабораторного проекта № 2, задание для которого дано ниже в подразд. 4.7.

**Верхнетреугольное разложение Холесского:**

---

$$P(N, N), \quad P = UU^T, \quad P > 0, \quad U — \text{ верхнетреугольная матрица.}$$
$$\text{DO } 5 \quad J = N, 2, -1 \quad \textcircled{C} \quad j = n, n - 1, \dots, 2$$
$$U(J, J) = \text{SQRT}(P(J, J))$$
$$\alpha = 1./U(J, J)$$
$$\text{DO } 5 \quad K = 1, J - 1$$
$$U(K, J) = \alpha * P(K, J)$$
$$\beta = U(K, J)$$
$$\text{DO } 5 \quad I = 1, K$$
$$5 \quad P(I, K) = P(I, K) - \beta * U(I, J)$$
$$U(1, 1) = \text{SQRT}(P(1, 1))$$

---

**Замечание 4.4.** Матрица  $U$  должна замещать  $P$  в компьютерной памяти. Нижние части матриц  $U$  и  $P$  не должны использоваться вовсе (память для них не выделяется). В любом случае верхнетреугольная часть матрицы  $P$  теряется, так как на её месте появляется верхнетреугольная часть матрицы  $U$ , т. е., все вычисления ведутся в одном и том же массиве  $P$ .

**Верхнетреугольное без  $\sqrt{\cdot}$  разложение Холесского:**

---

$$P(N, N), \quad P = \overline{U} D \overline{U}^T, \quad P > 0, \quad \overline{U} — \text{ верхнетреугольная матрица с единичной диагональю,} \quad D — \text{диагональная матрица.}$$
$$\text{DO } 5 \quad J = N, 2, -1 \quad \textcircled{C} \quad j = n, n - 1, \dots, 2$$
$$D(J) = P(J, J)$$
$$\alpha = 1./D(J)$$
$$\text{DO } 5 \quad K = 1, J - 1$$
$$\beta = P(K, J)$$
$$\overline{U}(K, J) = \alpha * \beta$$
$$\text{DO } 5 \quad I = 1, K$$
$$5 \quad P(I, K) = P(I, K) - \beta * \overline{U}(I, J)$$
$$D(1) = P(1, 1)$$

---

**Замечание 4.5.** В любом случае верхнетреугольная часть матрицы  $P$  теряется, так как на её месте появляется верхнетреугольная часть матрицы  $\overline{U}$ , при этом единицы, соответствующие диагонали матрицы  $\overline{U}$ , только подразумеваются, а на их месте пишутся диагональные элементы матрицы  $D$ . Для поддиагональной части массива  $P$  память не выделяется.

Предыдущие замечания **4.4** и **4.5** свидетельствуют, что фактически массив, выделяемый для исходной матрицы  $P$  и одновременно для получаемых на её месте результатов разложений Холесского, должен быть оформлен как одномерный массив. Размер этого массива, очевидно, равен  $N(N+1)/2$  элементов. Напишем для предыдущего алгоритма его «одномерную» версию.

**Верхнетреугольное без  $\sqrt{\cdot}$  «одномерное» разложение**  
 $P = \overline{U} D \overline{U}^T$ :

---

Одномерный массив  $P(N(N+1)/2)$  соответствует  $P = \overline{U} D \overline{U}^T$ .

---

$$JJ = N(N+1)/2$$

$$JJN = JJ$$

$$\text{DO } 5 \ J = N, 2, -1 \quad \textcircled{C} \quad j = n, n-1, \dots, 2$$

$$\alpha = 1./P(JJ)$$

$$KK = 0$$

$$JJN = JJ - J \quad \textcircled{C} \quad JJN = \text{следующий} \\ \text{диагональный элемент}$$

$$\text{DO } 4 \ K = 1, J-1$$

$$\beta = P(JJN + K) \quad \textcircled{C} \quad JJN + K = (K, J)$$

$$P(JJN + K) = \alpha * \beta$$

$$\text{DO } 3 \ I = 1, K$$

$$3 \ P(KK + I) = P(KK + I) - \beta * P(JJN + I) \quad \textcircled{C} \quad KK + I = (I, K)$$

$$4 \ KK = KK + K \quad \textcircled{C} \quad KK = K(K-1)/2$$

$$5 \ JJ = JJN \quad \textcircled{C} \quad JJ = J(J-1)/2$$


---

## 4.4 Разложение Холесского: $ijk$ -формы

Разложение Холесского симметричной положительно определённой матрицы  $P$  может быть получено в результате незначительных изменений базовых  $LU$ -разложений квадратной матрицы  $A$ . При этом симметрия матрицы  $P$  используется для сокращения числа действий примерно вдвое. Способ хранения матрицы  $P$  должен быть компактным, т. е., в одномерном массиве хранится по строкам (или по столбцам) только нижняя (или верхняя) треугольная часть матрицы  $P$  вместе с диагональю.

В той же последовательности, как выше изложены (см. подразд. 7.5)  $ijk$ -формы  $\overline{L} U$ -разложения матрицы  $A$ , приведём  $ijk$ -формы  $\overline{L} D \overline{L}^T$  и  $LL^T$ -разложений Холесского матрицы  $P > 0$ . Из них видно, сколь незначительны

требуемые изменения. В каждом алгоритме объединены оба разложения, при этом те изменения, что относятся к  $LL^T$ -разложению, заключены в скобки. В приводимых ниже алгоритмах явно не указано, когда элементы матриц  $D$ ,  $\bar{L}$  и  $L$  должны замещать соответствующие элементы исходной матрицы  $P$ . Такие замещения могут происходить сразу, а могут откладываться до того момента, когда элементы матрицы  $P$  станут ненужными для дальнейших вычислений. В этом отношении не все  $ijk$ -формы одинаково экономичны в реализации, и для каждой из них вопрос о скорейшем замещении исходных элементов матрицы  $P$  нужно решать отдельно.

### Два алгоритма Холесского: разложения $LL^T$ и $\bar{L}D\bar{L}^T$ с немедленными модификациями

#### 1) $kij$ -алгоритм

$$(l_{11} = p_{11}^{1/2})$$

Для  $k = 1$  до  $n - 1$

Для  $i = k + 1$  до  $n$

$$l_{ik} = p_{ik}/p_{kk}$$

$$(l_{ik} = p_{ik}/l_{kk})$$

Для  $j = k + 1$  до  $i$

$$p_{ij} = p_{ij} - l_{ik}p_{jk}$$

$$(p_{ij} = p_{ij} - l_{ik}l_{jk})$$

$$(l_{k+1,k+1} = p_{k+1,k+1}^{1/2})$$

#### 2) $kji$ -алгоритм

$$(l_{11} = p_{11}^{1/2})$$

Для  $k = 1$  до  $n - 1$

Для  $s = k + 1$  до  $n$

$$l_{sk} = p_{sk}/p_{kk} \quad (p_{sk}/l_{kk})$$

Для  $j = k + 1$  до  $n$

Для  $i = j$  до  $n$

$$p_{ij} = p_{ij} - l_{ik}p_{jk}$$

$$(p_{ij} = p_{ij} - l_{ik}l_{jk})$$

$$(l_{k+1,k+1} = p_{k+1,k+1}^{1/2})$$

### Четыре алгоритма Холесского: разложения $LL^T$ и $\bar{L}D\bar{L}^T$ с отложенными модификациями

#### 3) $jk i$ -алгоритм

$$(l_{11} = p_{11}^{1/2})$$

Для  $j = 2$  до  $n$

Для  $s = j$  до  $n$

$$l_{s,j-1} = p_{s,j-1}/p_{j-1,j-1}$$

$$(l_{s,j-1} = p_{s,j-1}/l_{j-1,j-1})$$

Для  $k = 1$  до  $j - 1$

Для  $i = j$  до  $n$

$$p_{ij} = p_{ij} - l_{ik}p_{jk}$$

$$(p_{ij} = p_{ij} - l_{ik}l_{jk})$$

$$(l_{j,j} = p_{j,j}^{1/2})$$

#### 4) $jik$ -алгоритм

$$(l_{11} = p_{11}^{1/2})$$

Для  $j = 2$  до  $n$

Для  $s = j$  до  $n$

$$l_{s,j-1} = p_{s,j-1}/p_{j-1,j-1}$$

$$(l_{s,j-1} = p_{s,j-1}/l_{j-1,j-1})$$

Для  $i = j$  до  $n$

Для  $k = 1$  до  $j - 1$

$$p_{ij} = p_{ij} - l_{ik}p_{jk}$$

$$(p_{ij} = p_{ij} - l_{ik}l_{jk})$$

$$(l_{j,j} = p_{j,j}^{1/2})$$

**5)  $ijk$ -алгоритм**

$$(l_{11} = p_{11}^{1/2})$$

Для  $i = 2$  до  $n$

Для  $k = 1$  до  $i - 1$

$$l_{i,k} = p_{i,k} / p_{k,k}$$

$$(l_{i,k} = p_{i,k} / l_{k,k})$$

Для  $j = k + 1$  до  $i$

$$p_{ij} = p_{ij} - l_{ik}p_{jk}$$

$$(p_{ij} = p_{ij} - l_{ik}l_{jk})$$

$$(l_{i,i} = p_{i,i}^{1/2})$$

**6)  $ijk$ -алгоритм**

$$(l_{11} = p_{11}^{1/2})$$

Для  $i = 2$  до  $n$

Для  $j = 2$  до  $i$

$$l_{i,j-1} = p_{i,j-1} / p_{j-1,j-1}$$

$$(l_{i,j-1} = p_{i,j-1} / l_{j-1,j-1})$$

Для  $k = 1$  до  $j - 1$

$$p_{ij} = p_{ij} - l_{ik}p_{jk}$$

$$(p_{ij} = p_{ij} - l_{ik}l_{jk})$$

$$(l_{i,i} = p_{i,i}^{1/2})$$

**Замечание 4.6.** Приведённые алгоритмы  $\overline{L} D \overline{L}^T$  и  $LL^T$ -разложений Холесского матрицы  $P$  получены из соответствующих  $ijk$ -алгоритмов  $\overline{U} U$ -разложения матрицы  $A$  (см. подразд. 7.5). Для получения  $\overline{U} D \overline{U}^T$  и  $UU^T$ -разложений Холесского матрицы удобно исходить из  $\overline{U} L$ -разложения матрицы  $A$ , если для него предварительно построить  $ijk$ -алгоритмы. Это построение нетрудно выполнить, если учесть, что  $\overline{U} L$ -разложение соответствует измененному (инверсному) порядку исключения переменных. В этом случае модификация системы уравнений начинается с последней переменной последнего уравнения.

Суммируя вышеизложенное по  $ijk$ -формам алгоритмов Холесского, полученных из  $ijk$ -форм алгоритмов Гаусса, имеем 24 разновидности разложений симметричной положительно определённой матрицы  $P$ :

6  $ijk$ -форм для  $P = \overline{L} D \overline{L}^T$ ,

6  $ijk$ -форм для  $P = LL^T$ ,

6  $ijk$ -форм для  $P = \overline{U} D \overline{U}^T$ ,

6  $ijk$ -форм для  $P = UU^T$ .

## 4.5 Разложение Холесского: алгоритмы окаймления

Как и для  $LU$ -разложения, для разложения Холесского в любых его вариантах (4.2) существует ещё один класс алгоритмов, — так называемые матрично-векторные алгоритмы, объединяемые идеей окаймления. Получение этих алгоритмов базируется на блочном перемножении матриц, участвующих в разложении. Здесь полностью применимы принципы, изложенные в подразд. 8.2.

Покажем, как выводятся такие матрично-векторные алгоритмы на примере одного из четырёх вариантов разложения Холесского (4.2), а именно, варианта  $P = LL^T$ . Пользуясь этим справочным материалом, любой студент сможет самостоятельно построить родственные алгоритмы для других трёх вариантов разложения. Для этого поделим все матрицы в данном варианте на блоки, выделяя в каждой матрице  $j$ -ю строку и  $j$ -й столбец. Тем самым разложение  $P = LL^T$  будет представлено в блочной форме

$$j \Rightarrow \begin{pmatrix} P_{11} & \overset{j}{\downarrow} & P_{13} \\ \mathbf{a}^T & \mathbf{a} & p_{jj} \\ P_{31} & \mathbf{b} & P_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & \overset{j}{\downarrow} & \\ \mathbf{c}^T & \mathbf{c} & l_{jj} \\ L_{31} & \mathbf{d} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11}^T & \overset{j}{\downarrow} & L_{31}^T \\ l_{jj} & \mathbf{d}^T & \\ & L_{33}^T & \end{pmatrix}, \quad (4.3)$$

где фрагменты  $j$ -й строки и  $j$ -го столбца обозначены как векторы-столбцы выделенными символами  $\mathbf{a}$ ,  $\mathbf{b}$ ,  $\mathbf{c}$  и  $\mathbf{d}$ , а заглавные буквы обозначают матрицы. Нулевые элементы треугольных матриц не показаны.

Перемножение матриц (4.3), выполняемое поблочно, дает девять соотношений относительно блок-элементов матриц  $P$  и  $L$ . Пользуясь этим, рассмотрим два основных способа разложения матрицы  $P$  методом окаймления.

**Окаймление известной части разложения.** Из указанных девяти соотношений возьмём только те, что окаймляют блок  $P_{11} = L_{11}L_{11}^T$ , считая, что в этой части разложение уже сделано, т. е., что блок  $L_{11}$  уже вычислен. В силу симметрии  $P$  из трёх окаймляющих произведений имеем только два:

$$\mathbf{a} = L_{11}\mathbf{c} \quad \text{и} \quad p_{jj} = \mathbf{c}^T\mathbf{c} + l_{jj}^2. \quad (4.4)$$

Отсюда сначала находим  $\mathbf{c}$  как решение нижнетреугольной системы уравнений  $L_{11}\mathbf{c} = \mathbf{a}$ ; затем находим  $l_{jj} = (p_{jj} - \mathbf{c}^T\mathbf{c})^{1/2}$ .

**Окаймление неизвестной части разложения.** Из указанных соотношений возьмём те, что окаймляют блок  $P_{33}$ , считая, что до этого блока разложение уже сделано, т. е., что блоки  $L_{11}$ ,  $L_{31}$  и  $\mathbf{c}$  уже найдены. В силу симметрии  $P$  из трёх окаймляющих произведений имеем только два:

$$p_{jj} = \mathbf{c}^T\mathbf{c} + l_{jj}^2 \quad \text{и} \quad \mathbf{b} = L_{31}\mathbf{c} + \mathbf{d}l_{jj}. \quad (4.5)$$

Отсюда сначала находим  $l_{jj} = (p_{jj} - \mathbf{c}^T\mathbf{c})^{1/2}$ ; затем  $\mathbf{d} = (\mathbf{b} - L_{31}\mathbf{c})/l_{jj}$ .

Существует два естественных способа реализации окаймления известной части в  $LL^T$ -разложении.

В первом варианте треугольная система в (4.4) решается с помощью строчного алгоритма (аналог алгоритма на рис. 8.1 слева), во втором — с помощью алгоритма скалярных произведений (аналог алгоритма на рис. 8.1 справа). Псевдокоды этих двух вариантов приведены на рис. 4.1.

$l_{11} = \sqrt{p_{11}}$	$l_{11} = \sqrt{p_{11}}$
Для $j = 2$ до $n$	Для $j = 2$ до $n$
Для $k = 1$ до $j - 1$	Для $i = 2$ до $j$
$l_{jk} = p_{jk}/l_{kk}$	$l_{j,i-1} = p_{j,i-1}/l_{i-1,i-1}$
Для $i = k + 1$ до $j$	Для $k = 1$ до $i - 1$
$p_{ji} = p_{ji} - l_{jk}l_{ik}$	$p_{ji} = p_{ji} - l_{j,k}l_{ik}$
$l_{jj} = \sqrt{p_{jj}}$	$l_{jj} = \sqrt{p_{jj}}$

Рис. 4.1. Алгоритмы окаймления известной части  $LL^T$ -разложения: строчный (слева); алгоритм скалярных произведений (справа)

Для окаймления неизвестной части в  $LL^T$ -разложении также существуют два естественных способа реализации выражений (4.5). Здесь основной операцией является умножение вектора на прямоугольную матрицу.

Можно реализовать такие умножения посредством скалярных произведений или линейных комбинаций, что приводит к двум различным формам алгоритма, показанным на рис. 4.2, которые аналогичны алгоритмам Донгарры–Айзенштата на рис. 8.4.

Для $j = 1$ до $n$	Для $j = 1$ до $n$
Для $k = 1$ до $j - 1$	Для $i = j + 1$ до $n$
$p_{jj} = p_{jj} - l_{jk}l_{jk}$	Для $k = 1$ до $j - 1$
$l_{jj} = \sqrt{p_{jj}}$	$p_{ij} = p_{ij} - l_{ik}l_{jk}$
Для $k = 1$ до $j - 1$	Для $k = 1$ до $j - 1$
Для $i = j + 1$ до $n$	$p_{jj} = p_{jj} - l_{jk}l_{jk}$
$p_{ij} = p_{ij} - l_{ik}l_{jk}$	$l_{jj} = \sqrt{p_{jj}}$
Для $s = j + 1$ до $n$	Для $s = j + 1$ до $n$
$l_{sj} = p_{sj}/l_{jj}$	$l_{sj} = p_{sj}/l_{jj}$

Рис. 4.2. Алгоритмы окаймления неизвестной части  $LL^T$ -разложения: алгоритм линейных комбинаций (слева); алгоритм скалярных произведений (справа)

Таким образом, выше показано, что алгоритмы окаймления в  $LU$ -разложении (см. разд. 8.1) легко модифицируются для случая симметрической

положительно определённой матрицы  $P$ . Тогда мы имеем 4 варианта разложения Холесского (4.2), 2 способа окаймления и 2 схемы вычислений для каждого алгоритма окаймления. Всего получается 16 вариантов алгоритмов окаймления для разложения Холесского симметрической положительно определённой матрицы. Добавляя к ним 24 разновидности  $ijk$ -форм, получаем 40 различных вычислительных схем разложений Холесского, которые и составляют весь набор вариантов (см. подразд. 4.8) задания на лабораторный проект № 2 (см. подразд. 4.7).

## 4.6 Особенности хранения ПО-матрицы $P$

Как уже отмечалось (см. стр. 141), особенностью данного проекта является использование *линейных (одномерных)* массивов для хранения матрицы  $P$ . Так как матрица  $P$  симметрическая, то достаточно хранить только нижнюю (или верхнюю) треугольную часть этой матрицы вместе с диагональю. Причём для хранения заполненной матрицы используется один одномерный массив, а для хранения разреженной – два.

Хранение матрицы  $P$  может быть организовано по столбцам или по строкам в зависимости от используемого алгоритма разложения.

Рассмотрим строчный вариант хранения нижней треугольной части заполненной матрицы  $P$ . В этом случае все элементы нижней треугольной матрицы записываются построчно в одномерный массив. Так как для хранения первой строки матрицы требуется один элемент массива, для хранения второй строки – два элемента и т. д., то для хранения симметрической матрицы размера  $n$  требуется одномерный массив размера  $n(n + 1)/2$ . Положение  $(i, j)$ -го элемента матрицы  $P$  в массиве определяется по формуле

$$k = (i - 1)i/2 + j.$$

Аналогичным образом организуется хранение матрицы  $P$  по столбцам.

Как уже отмечалось, для хранения разреженной матрицы  $P$  используются два одномерных массива. Так, хранение нижней треугольной части матрицы  $P$  по строкам можно организовать следующим образом. В массиве  $\mathbf{a}$  хранятся построчно элементы матрицы от первого ненулевого до диагонального включительно. В массиве  $\mathbf{b}$  на  $i$ -м месте стоит положение  $i$ -го диагонального элемента матрицы  $P$  в массиве  $\mathbf{a}$ . Для определения положения  $(i, j)$ -го элемента матрицы  $P$  в массиве  $\mathbf{a}$  надо воспользоваться следующим алгоритмом. Сначала вычисляем  $k = b(i) - (i - j)$ . Затем, если  $k > b(i - 1)$ , то этот элемент стоит на

$k$ -м месте. В противном случае  $(i, j)$ -й элемент стоит левее первого ненулевого элемента  $i$ -й строки, поэтому он равен нулю и в массиве  $a$  не хранится. Способ хранения по столбцам строится аналогичным образом, но в этом случае хранятся все элементы от диагонального до последнего ненулевого элемента столбца включительно.

Таким образом, существуют 4 варианта хранения разреженной ленточной матрицы  $P$ , и выбор конкретного варианта должен соответствовать заданному варианту разложения Холесского и разновидности  $ijk$ -форм.

**Замечание 4.7.** С учётом положительной определённости матриц этой лабораторной работы, процедура выбора главного элемента, а также процедуры перестановки строк и столбцов матрицы  $P$  отсутствуют как для заполненных, так и для разреженных матриц.

## 4.7 Задание на лабораторный проект № 2

Написать и отладить программу, реализующую заданный вариант алгоритма разложения Холесского, для решения системы  $Px = f$ , где  $P$  – симметрическая положительно определённая матрица (ПО-матрица  $P$ , или кратко, матрица  $P > 0$ ). Отделить основные части программы:

- а) подпрограмму генерации ПО-матрицы  $P$ ;
- б) подпрограмму разложения Холесского;
- в) подпрограмму решения систем линейных алгебраических уравнений;
- г) подпрограмму решения систем линейных алгебраических уравнений с ленточной матрицей  $P$ ;
- д) сервисные подпрограммы, включая демонстрацию разложения на экране, подпрограмму контроля правильности разложения и др.

Уделить особое внимание эффективности программы (в смысле экономии оперативной памяти). Для этого в одномерном массиве достаточно хранить только нижнюю (или верхнюю) треугольную часть и диагональ матрицы  $P$ . Результат разложения замещает исходную матрицу. Предусмотреть пошаговое выполнение алгоритма исключения с выводом результата на экран.

Выполнить следующие пункты задания:

1. Дать формулировку и доказательство теоремы о заданном варианте разложения Холесского как утверждение о том, что предлагаемый студентом алгоритм даёт единственное разложение Холесского. Для иллюстрации дать численный пример работы алгоритма по шагам для матрицы  $P$  размера  $(4 \times 4)$ .

2. Провести подсчёт количества операций:

- а) извлечения квадратного корня;
- б) умножения и деления.

Подсчёт выполнить тремя способами:

- а) фактически — при конкретном расчёте разложения;
- б) теоретически точно в зависимости от размерности матрицы  $n$ ;
- в) теоретически приближенно в зависимости от  $n$  при  $n \rightarrow \infty$ .

3. Определить скорость решения задач (решение систем линейных алгебраических уравнений), для чего спроектировать и провести эксперимент, который охватывает сгенерированные случайным образом ПО-матрицы  $P$  порядка от 5 до 100 (через 5 порядков). Результаты представить в виде таблицы и графика зависимости времени выполнения от порядка матриц. Сравнить со своим вариантом из лабораторного проекта № 1.

4. Оценить точность решения систем линейных алгебраических уравнений с матрицами из п. 3. Для этого выбрать точное решение  $x^*$  и образовать правые части  $f = Px^*$ . В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)$ . Для оценки точности использовать норму вектора (3.20), с. 110 из лабораторного проекта № 1. Провести анализ точности вычисленного решения  $x$  в зависимости от порядка матрицы. Результаты представить в виде таблицы и графика. Сравнить со своим вариантом из проекта № 1.

5. Для заполнения матрицы  $P$  использовать случайные числа из диапазона от  $-100$  до  $100$ . Сначала заполнить треугольную часть матрицы  $P$ , т. е., элементы  $p_{ij}$ , где  $i > j$ . Затем заполнить диагональ. В качестве диагонального элемента  $p_{ii}$ ,  $i = 1, 2, \dots, n$ , выбрать случайное число из интервала

$$\left[ \sum_{j \neq i} |p_{ij}| + 1, \sum_{j \neq i} |p_{ij}| + 101 \right], \quad (4.6)$$

чтобы обеспечить выполнение условия

$$p_{ii} \geq \sum_{j \neq i} |p_{ij}| + 1,$$

гарантирующего положительную определённость матрицы  $P$ .

6. Определить скорость и точность решения систем линейных алгебраических уравнений с разреженными ленточными матрицами. Для этого спроектировать и провести эксперимент для систем порядка от 100 до 200 (через 5). Результаты представить в виде таблиц и графиков зависимости скорости и точности решения от порядка матриц. Для этих же систем найти аналогичные зависимости для обычного метода Холесского. Результаты сравнить.

7. Для случайного заполнения разреженной ленточной матрицы  $P$  использовать следующий алгоритм:

- случайным образом заполнить половину матрицы (верхнюю или нижнюю), включая диагональ;
- в случае заполнения нижней треугольной части матрицы  $P$  в  $i$ -й строке,  $i = 1, 2, \dots, n$ , случайным образом определить количество ненулевых элементов (от 1 до 10), их местоположение (номер столбца от  $\max\{1, i - 50\}$  до  $i - 1$ ) и значение (ненулевые целые числа, лежащие в интервале от  $-100$  до  $100$ );
- при заполнении верхней треугольной части матрицы  $P$  применять тот же алгоритм, что и в п. б), с той лишь разницей, что номер столбца лежит в интервале от  $i + 1$  до  $\min\{i + 50, n\}$ ;
- диагональный элемент в  $i$ -й строке,  $i = 1, 2, \dots, n$ , определить случайным образом на интервале (4.6).

В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)$ ,  $n$  – порядок матрицы  $P$ . В случае, если при решении системы  $Px = f$  выяснится, что матрица  $P$  вырождена (плохо обусловлена), то сгенерировать новую матрицу того же порядка и решить систему линейных алгебраических уравнений с новой матрицей  $P$  и новой правой частью. Для оценки точности решения использовать норму вектора (3.20), с. 110 из лабораторного проекта № 1.

**Замечание 4.8.** По ходу проведения всех численных экспериментов на экран дисплея должны выводиться следующие таблицы.

#### Число вычислительных операций:

Порядок матрицы	Квадратные корни			Умножение и деление		
	а	б	с	а	б	в

где а, б, в означает способ вычисления числа действий (см. п. 2).

**Решение систем линейных алгебраических уравнений  
с заполненной матрицей  $P$ :**

Порядок матрицы	Время		Точность	
	метод Гаусса	метод Холесского	метод Гаусса	метод Холесского

Таким образом, в данный проект следует включить работу, выполненную ранее в проекте № 1, чтобы иметь возможность сравнения метода Холесского с методом Гаусса как по времени, так и по точности вычислений.

**Решение систем линейных алгебраических уравнений  
с разреженной матрицей  $P$ :**

Порядок матрицы	Время		Точность	
	Заполненная матрица	Разреженная матрица	Заполненная матрица	Разреженная матрица

Это означает, что для каждого текущего значения  $n$  порядка матрицы  $P$  необходимо решать две системы: одну – с заполненной матрицей  $P$  (см. п. 5 задания), другую – с разреженной матрицей  $P$  (см. п. 7 задания).

**Замечание 4.9.** Необходимо вывести на экран следующие графики:

**Графики решения систем с заполненной матрицей  $P$ :**

- зависимость времени решения от порядка матриц для методов Гаусса и Холесского;
- зависимость точности решения от порядка матриц для методов Гаусса и Холесского.

**Графики решения систем с разреженной матрицей  $P$ :**

- зависимость времени решения от порядка матриц для обычного метода Холесского и с учётом разреженности матрицы  $P$ ;
- зависимость точности решения от порядка матриц для обычного метода Холесского и с учётом разреженности матрицы  $P$ . При построении графиков использовать данные из соответствующей таблицы. Для этого их необходимо записать в текстовый файл.

## 4.8 Варианты задания на лабораторный проект № 2

Как отмечалось в конце подразд. 4.5, всего по данной теме предлагается 40 различных вычислительных схем разложений Холесского, которые и составляют весь набор вариантов задания на лабораторный проект № 2 (см. подразд. 4.7)

В табл. 4.1 каждому номеру варианта соответствует своя разновидность разложения Холесского и свой способ организации вычислений.

Таблица 4.1. Варианты задания на лабораторный проект № 2

Вид разложения	ijk-формы						Окаймление			
	$kij$	$kji$	$jki$	$jik$	$ikj$	$ijk$	известной части		неизвестной части	
							a	b	c	b
$P = \bar{L} D \bar{L}^T$	1	2	3	4	5	6	7	8	9	10
$P = LL^T$	11	12	13	14	15	16	17	18	19	20
$P = \bar{U} D \bar{U}^T$	21	22	23	24	25	26	27	28	29	30
$P = UU^T$	31	32	33	34	35	36	37	36	39	40

<sup>a</sup> – строчный алгоритм;

<sup>b</sup> – алгоритм скалярных произведений;

<sup>c</sup> – алгоритм линейных комбинаций.

Если нет других указаний преподавателя, выбирайте ваш вариант по вашему номеру в журнале студенческой группы.

## 4.9 Методические рекомендации для проекта № 2

Рассмотрим следующие пункты задания последовательно с точки зрения их реализации в проекте.

- Система меню для взаимодействия пользователя с программой.
- Функция генерации ПО-матрицы  $P$ .
- Функция разложения Холесского.
- Функция решения системы линейных алгебраических уравнений.

5. Функция решения системы линейных алгебраических уравнений с ленточной матрицей  $P$ . Ленточную матрицу мы ниже называем также разреженной.
6. Эксперимент 1 «Количество арифметических операций».
7. Эксперимент 2 «Решение СЛАУ с заполненной матрицей  $P$ ».
8. Эксперимент 3 «Решение СЛАУ с разреженной матрицей  $P$ ».

С учётом положительной определённости матрицы  $P$  процедура выбора главного элемента, а также процедуры перестановки строк и столбцов матрицы  $P$  отсутствуют как для заполненных, так и для разреженных матриц.

### **Система меню для взаимодействия пользователя с программой**

Действуйте так же, как вы строили меню на с. 115 при реализации п. 1 из перечня заданий для проекта № 1 на с. 114.

Необходимо также реализовать следующие сервисные подпрограммы: демонстрация разложения на экране, подпрограмма контроля правильности разложения.

### **Функция генерации ПО-матрицы $P$**

Для заполнения матрицы  $P$  нужно задать случайные числа из диапазона от  $-100$  до  $+100$ . Поскольку матрица  $P$  симметрическая, достаточно хранить только нижнюю (или верхнюю) треугольную часть этой матрицы вместе с диагональю.

### **Заполненная матрица $P$**

Для строчного хранения заполненной симметрической матрицы размера  $n$  требуется одномерный массив размера  $n(n + 1)/2$ . Хранить элементы можно как по строкам, так и по столбцам. Положение  $(i, j)$ -го элемента матрицы  $P$  в массиве можно определить по формуле  $k = (i - 1)i/2 + j$ .

Для генерации ПО-матрицы  $P$  необходимо:

1. Заполнить треугольную часть матрицы  $P$ , т. е., элементы  $p_{ij}$ , где  $i > j$ , случайными числами из диапазона от  $-100$  до  $+100$ .

2. Заполнить диагональ. В качестве диагонального элемента  $p_{ii}$ ,  $1 \leq i \leq n$ , нужно выбрать случайное число из интервала

$$\left[ \sum_{j \neq i} |p_{ij}| + 1, \quad \sum_{j \neq i} |p_{ij}| + 101 \right].$$

Рассмотрим один из возможных вариантов генерации заполненной матрицы  $P$  (для хранения по строкам):

### Листинг 4.1.

```
// генерируем внедиагональные элементы матрицы (n - размер матрицы),
// вычисляем сумму элементов матрицы для j!=i
// вычисляем сумму элементов матрицы для j,i
int [] sum= new int[n];
Random r= new Random();
for (int j=1;j<n-1;j++)
    for (int i=(j+1);i<n;i++) {
        A[i,j]=r.Next(-100,100);
        A[j,i]=A[i,j];
    }
for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
        if (i!=j)
            sum[i]=sum[i]+Math.Abs(A[i,j]);

// выделяем память под массив для хранения матрицы P
int [] P= new int[(n*(n+1)/2)];
int ch=0; // счётчик

// Заполняем массив для хранения нижней треугольной части
// матрицы P по строкам:

for (int i=0;i<n;i++){
    for (int j=1;j<=i;j++){
        if (i==j) // если элемент диагональный, то
            P[ch] = r.Next(sum[i]+1,sum[i]+102);
            // выбираем случайное число из интервала
        else
            P[ch] = r.Next(0,100);
    }
}
// Вычисляем элементы верхней треугольной части матрицы P
for (int i=0;i<n;i++)
    for (int j=i+1;j<n;j++)
        P[i,j]=P[j,i];
```

```

P [ch] = A [i , j] ;
ch++ ;
}
}

```



## Генерации ПО-матрицы $P$ , если матрица разреженная

Для хранения разреженной матрицы  $P$  потребуется два одномерных массива. Хранить элементы можно как по строкам, так и по столбцам. При строчном варианте хранения в массиве  $a$  хранятся построчно элементы матрицы от первого ненулевого до диагонального включительно. В массиве  $b$  на  $i$ -м месте стоит положение  $i$ -го диагонального элемента матрицы  $P$  в массиве  $A$ .

Для случайного заполнения нижней или верхней треугольной части разреженной ленточной матрицы  $P$  необходимо использовать следующий алгоритм:

- в случае заполнения нижней треугольной части матрицы  $P$  в  $i$ -й строке,  $i = 2, 3, \dots, n$ , случайным образом определить количество ненулевых элементов (от 1 до 10), их местоположение (номер столбца от  $\max 1, i - 50$  до  $i - 1$ ) и значение (ненулевые целые числа, лежащие в интервале от  $-100$  до  $+100$ );
- при заполнении верхней треугольной части матрицы  $P$  применять тот же алгоритм, что и в п. а), с той лишь разницей, что номер столбца лежит в интервале от  $i + 1$  до  $\min i + 50, n$ ;
- диагональный элемент в  $i$ -й строке,  $i = 1, 2, \dots, n$ , определить случайным образом на интервале

$$\left[ \sum_{j \neq i} |p_{ij}| + 1, \quad \sum_{j \neq i} |p_{ij}| + 101 \right].$$

Положение  $(i, j)$ -го элемента матрицы  $P$  в массиве  $a$  можно определить по следующему алгоритму:

- Сначала вычисляем  $k = b(i) - (i - j)$ .
- Затем, если  $k > b(i - 1)$ , то этот элемент стоит на  $k$ -м месте.

3. В противном случае  $(i, j)$ -й элемент стоит левее первого ненулевого элемента  $i$ -й строки, поэтому он равен нулю и в массиве  $a$  не хранится.

Способ хранения по столбцам строится аналогичным образом, но в этом случае следует хранить все элементы от диагонального до последнего ненулевого элемента столбца включительно.

## Функция разложения Холлесского

Подробное описание алгоритмов Холлесского можно найти в подразд. 4.4 и подразд. 4.5.

В лабораторном проекте все вычисления должны проводиться в одном и том же массиве. Это значит, что элементы матрицы  $P$  должны замещаться элементами матриц  $L$ ,  $U$  или  $D$  (в зависимости от варианта разложения) по мере вычисления последних. Замещения могут происходить сразу, а могут откладываться до того момента, когда элементы матрицы  $P$  станут не нужны для дальнейших вычислений.

Рассмотрим реализацию на примере  $kij$ -алгоритма разложения  $\bar{L} D \bar{L}^T$  (см. подразд. 4.4):

$\text{Для } k = 1 \text{ до } n - 1$ $\quad \text{Для } i = k + 1 \text{ до } n$ $\quad \quad l_{ik} = p_{ik}/p_{kk}$ $\quad \quad \text{Для } j = k + 1 \text{ до } i$ $\quad \quad \quad p_{ij} = p_{ij} - l_{ik}p_{jk}$
---

Рассмотрим один из возможных способов реализации данного алгоритма. Пусть в одномерном массиве  $P$  хранится нижняя треугольная часть ПО-матрицы  $P$ . Замещение элементов матрицы  $P$  элементами матрицы  $\bar{L}$  будет происходить сразу при их вычислении. Заменим 3-ю строку  $l_{ik} = p_{ik}/p_{kk}$  на  $p_{ik} = p_{ik}/p_{kk}$ . Соответственно, заменим 5-ю строку, подставив  $p_{ik}$  вместо  $l_{ik}$  и умножив  $p_{jk}$  на  $p_{kk}$ :

$$p_{ij} = p_{ij} - l_{ik}p_{jk} = p_{ij} - p_{ik}p_{jk}p_{kk}.$$

Для обращения к  $(i, j)$ -му элементу будем пользоваться формулой:

$$k = (i - 1)i/2 + j.$$

**Листинг 4.2.**

```
for (int k=0;k<(n-1);k++) {  
    for (int i=(k+1);i<n;i++) {  
        int ik=(int)((i-1)*i/2)+k;  
        int kk=(int)((k-1)*k/2)+k;  
        P[ik]=P[ik]/P[kk];  
        for (int j=(k+1);j<=i;j++) {  
            int ij=(int)((i-1)*i/2)+j;  
            int jk=(int)((j-1)*j/2)+k;  
            P[ij]=P[ij]-P[ik]*P[jk]*P[kk];  
        }  
    }  
}
```



В результате выполнения алгоритма получим нижнюю треугольную матрицу  $P$  с положительными элементами на диагонали. Матрицу  $\bar{L}$  найдём из получившейся матрицы  $P$ , заменив все диагональные элементы единицами, диагональную матрицу  $D$  – заполнив её диагональ диагональными элементами матрицы  $P$ .

**Функция решения системы линейных алгебраических уравнений**

Рассмотрим детали реализации алгоритма нахождения решения СЛАУ по разложению Холесского. Решение будем искать в два этапа. Но сначала приведём математическое обоснование для варианта  $\bar{L} \bar{D} \bar{L}^T$ -разложения:

$$\begin{aligned} Px = f &\Rightarrow \bar{L} \bar{D} \bar{L}^T x = f \Rightarrow y \triangleq \bar{D} \bar{L}^T x \Rightarrow \bar{L} y = f \\ &\Rightarrow z \triangleq \bar{L}^T x \Rightarrow Dz = y \Rightarrow \bar{L}^T x = z \Rightarrow x. \end{aligned}$$

Аналогичная цепочка вычислений получается и для других видов разложения. Рассмотрим реализацию алгоритма решения СЛАУ на примере  $\bar{L} \bar{D} \bar{L}^T$ -разложения.

**Первый проход:** известны матрица  $\bar{L}$  и вектор  $f$ , находим неизвестный вектор  $y$  с помощью прямой подстановки:

**Листинг 4.3.**

```
// метод прямой подстановки (скалярных произведений)
for (int i=0;i<n;i++){
    float sum=0;
    if (i>1){
        for (int j=1;j<=i-1;j++){
            int ij=(int)((i-1)*i/2)+j;
            sum=sum+L[ij]*y[j];
        }
    }
    y[i]=f[i]-sum;
}
```



**Второй проход:** рассмотрим произведение матриц  $Dz = y$ . Известны вектор  $y$  и матрица  $D$ . Находим неизвестный вектор  $z$ :

**Листинг 4.4.**

```
for (int k=0;k<n;k++){
    int kk=(int)((k-1)*k/2)+k;
    z[k]=y[k]/P[kk];
}
// Найденное решение находится в массиве z
```



**Третий проход:** известны матрица  $\bar{L}$  и вектор  $z$  (помещённый в массив  $z$  в листинге 4.4), находим искомый вектор  $x$  (помещённый в массив  $x$  в листинге 4.5) с помощью обратной подстановки:

**Листинг 4.5.**

```
// метод обратной подстановки (скалярных произведений)
for (int i=n-1;i>-1;i--){
    float sum=0;
    if (i<n) {
        for (int j=i+1;j<n;j++) {
            int ij=(int)((i-1)*i/2)+j;
            sum =sum+P[ij]*x[j];
        }
    }
    x[i]=sum;
}
```

```
    }
}
x[i]=z[i]-sum;
}
// Найденное решение находится в массиве x
```



## Функция решения системы линейных алгебраических уравнений с ленточной матрицей $P$

Для решения СЛАУ с ленточной матрицей можно воспользоваться алгоритмом, указанным в пункте на с. 156. Для определения положения элемента в одномерном массиве воспользоваться алгоритмом, описанным в пункте на с. 154.

### Эксперимент 1 «Количество арифметических операций»

Написать реализацию первого эксперимента для исследования количества арифметических операций. Описание эксперимента можно найти в подразд. 4.7, с. 147.

#### Методика проведения эксперимента:

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

#### Число вычислительных операций:

Порядок матрицы	Квадратные корни			Умножение и деление		
	а	б	в	а	б	в

где а, б, в означает способ вычисления числа действий: а – фактическое, б – теоретически точное, в – теоретически приближенное. Каждая строка в таблице будет зависеть от  $n$  – порядка матрицы. Число  $n$  должно изменяться от 5 до 100 через 5 порядков, т. е., в таблице будет 20 строк.

2. Присвоить  $n$  очередное значение.
3. Сгенерировать матрицу  $P$  в соответствии с п. 2 списка заданий на с. 151.

4. Выполнить разложение матрицы  $P$  в соответствии с п. 3 списка заданий на с. 151.
5. Подсчитать фактическое число операций извлечения квадратного корня, умножения и деления с помощью специальных счётчиков, которые вы добавите в функцию разложения.
6. Подсчитать теоретически точное число операций извлечения квадратного корня (равное  $n$  – размерности ПО-матрицы  $P$ ), операций умножения и деления. Например, для  $\overline{L} \overline{D} \overline{L}^T$ -разложения докажите, что оно равно

$$\sum_{i=1}^n ((n-i)(i-1) + (i-1) + 1)$$

в зависимости от размерности матрицы  $n$ .

7. Подсчитать теоретически приближенное число операций извлечения квадратного корня (равное  $n$  – размерности ПО-матрицы  $P$ ), операций умножения и деления (равное  $n^3/6$ ) в зависимости от  $n$  при  $n \rightarrow \infty$ .
8. Добавить в таблицу полученные экспериментальные данные: порядок  $n$ , фактическое, теоретически точное и теоретически приближенное число операций извлечения квадратного корня, умножения и деления.

## Эксперимент 2 «Решение СЛАУ с заполненной матрицей $P$ »

Реализовать второй эксперимент для исследования скорости и погрешности решения СЛАУ в случае, когда матрица  $P$  является заполненной. Описание эксперимента можно найти в подразд. 4.7, с. 147.

Перед проведением эксперимента необходимо написать процедуру, вычисляющую погрешность решения СЛАУ. В данный проект следует включить программный код, созданный ранее в проекте № 1, для того чтобы иметь возможность сравнить метод Холесского с методом Гаусса по времени выполнения и по погрешности вычислений.

### **Методика проведения эксперимента:**

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

**Решение СЛАУ с заполненной матрицей  $P$** 

Порядок матрицы	Время		Погрешность	
	метод Гаусса	метод Холесского	метод Гаусса	метод Холесского

Каждая строка в таблице будет зависеть от  $n$  – порядка матрицы. Число  $n$  должно изменяться от 5 до 100 через 5 порядков, т. е., в таблице будет 20 строк.

2. Присвоить  $n$  очередное значение.
3. Заполнить матрицу  $P$  случайными числами в соответствии с п. 2 списка заданий на с. 151. Сгенерировать вектор  $f$  (по правилу ниже, п. 7).
4. Выполнить разложение матрицы  $P$  в соответствии с п. 3 списка заданий на с. 151.
5. Найти решение СЛАУ  $x$  в соответствии с п. 4 списка заданий на с. 151.
6. Подсчитать скорость решения задачи как сумму времени разложения матрицы и времени решения СЛАУ.
7. Оценить погрешность решения СЛАУ:
  - Задать точное решение  $x^*$ . В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – порядок матрицы.
  - Образовать правые части  $f := Px^*$ , матрица  $P$  известна из п. 3.
  - Найти решение СЛАУ  $Px = f$ .
  - Вычислить погрешность решения СЛАУ как максимальный модуль разности компонента вектора точного решения и вектора найденного решения.
8. Добавить в таблицу полученные экспериментальные данные для разложения Холесского и для метода Гаусса (использовать данные лабораторного проекта № 1): порядок  $n$ , время выполнения, погрешность решения СЛАУ.
9. Записать в файл полученные экспериментальные данные для разложения Холесского: порядок  $n$ , время выполнения, погрешность решения СЛАУ.
10. Пункты 2–9 повторить для всех значений  $n$ .

11. Построить графики решения СЛАУ с заполненной матрицей  $P$ :
  - зависимость времени решения от порядка матриц для методов Гаусса и Холесского;
  - зависимость погрешности решения от порядка матриц для методов Гаусса и Холесского.
12. Сравнить полученные данные со своим вариантом из лабораторного проекта № 1.
13. Проанализировать полученные данные и сделать содержательные выводы об эффективности и качестве реализованных численных методов.

### **Эксперимент 3 «Решение СЛАУ с разреженной матрицей $P$ »**

Написать реализацию третьего эксперимента для исследования скорости и погрешности решения СЛАУ в случае, когда матрица  $P$  является разреженной. Описание эксперимента можно найти в подразд. 4.7, с. 147. Перед проведением эксперимента необходимо написать процедуру, вычисляющую погрешность решения СЛАУ (взять из лабораторного проекта № 1).

#### **Методика проведения эксперимента:**

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

#### **Решение СЛАУ с разреженной матрицей $P$**

Порядок матрицы	Время		Погрешность	
	Заполненная матрица	Разреженная матрица	Заполненная матрица	Разреженная матрица

Для каждого текущего значения  $n$  порядка матрицы  $P$  необходимо решать две системы: одну – с заполненной матрицей  $P$  в соответствии с п. 4 списка заданий на с. 151, другую – с разреженной матрицей  $P$  в соответствии с п. 5 списка заданий на с. 151. Каждая строка в таблице будет зависеть от  $n$  – количества уравнений в системе. Число  $n$  должно изменяться от 100 до 200 через 5 порядков, т. е., в таблице будет 21 строка.

2. Присвоить  $n$  очередное значение. Заполнить случайными числами заполненную матрицу  $P$  в соответствии с п. 2 списка заданий на с. 151. Сгенерировать вектор  $f$  (по правилу ниже, п. 13).

3. Выполнить разложение матрицы  $P$  в соответствии с п. 3 списка заданий на с. 151.
4. Найти решение СЛАУ  $x$  в соответствии с п. 4 списка заданий на с. 151.
5. Подсчитать скорость решения задачи так же, как и в эксперименте 2 с заполненной матрицей.
6. Оценить погрешность решения СЛАУ так же, как и в эксперименте 2.
7. Добавить в таблицу полученные экспериментальные данные: порядок  $n$ , время выполнения, погрешность решения СЛАУ.
8. Записать в файл полученные экспериментальные данные: порядок  $n$ , время выполнения, погрешность решения СЛАУ.
9. Заполнить разреженную ленточную матрицу  $P$  случайными числами в соответствии с п. 2 списка заданий на с. 151.
10. Выполнить разложение матрицы  $P$  в соответствии с п. 3 списка заданий на с. 151.
11. Найти решение СЛАУ  $x$  в соответствии с п. 4 списка заданий на с. 151.
12. Подсчитать скорость решения задачи так же, как и в эксперименте 2 с заполненной матрицей.
13. Оценить погрешность решения СЛАУ:
  - Задать точное решение  $x^*$ . В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – порядок матрицы.
  - Вычислить правые части  $f := Px^*$ .
  - Найти решение СЛАУ  $Px = f$ . В случае, если при решении системы  $Px = f$  выяснится, что матрица  $P$  вырождена (плохо обусловлена), то сгенерировать новую матрицу того же порядка и решить систему линейных алгебраических уравнений с новой матрицей  $P$  и новой правой частью  $f$ . Теоретически, этого не должно случиться в силу правила генерации матрицы  $P$ , которое гарантирует свойство  $P > 0$ .
  - Вычислить погрешность решения СЛАУ как максимальный модуль разности компонентов вектора точного решения и вектора найденного решения.

14. Добавить в таблицу полученные экспериментальные данные: порядок  $n$ , время выполнения, погрешность решения СЛАУ.
15. Записать в файл полученные экспериментальные данные: порядок  $n$ , время выполнения, погрешность решения СЛАУ.
16. Пункты 2–15 повторить для всех значений  $n$ .
17. Построить графики решения СЛАУ с разреженной матрицей  $P$ :
  - зависимость времени решения от порядка матриц для обычного метода Холесского и для метода с учётом разреженности матрицы  $P$ ;
  - зависимость погрешности решения от порядка матриц для обычного метода Холесского и для метода с учётом разреженности матрицы  $P$ .

При построении графиков использовать данные из файла.

18. Проанализировать полученные данные и сделать содержательные выводы об эффективности и качестве реализованных численных методов.

## 4.10 Тестовые задачи для проекта № 2

Используйте приводимые ниже задачи в двух режимах:

- для контроля собственного понимания алгоритма,
- для контроля правильности вашего программирования.

### Задача 1

*Для матрицы*

$$P = \begin{pmatrix} 4 & -2 & 2 & 4 \\ -2 & 2 & -3 & 3 \\ 2 & -3 & 14 & -8 \\ 4 & 3 & -8 & 33 \end{pmatrix}$$

*выполнить следующее:*

- a. Построить  $LL^T$ -разложение матрицы  $P$  ( $L$  – нижняя треугольная матрица с положительными элементами диагонали).
- б. С помощью  $LL^T$ -разложения матрицы  $P$  решить систему

$$Px = b,$$

с вектором  $b = (4, -10, 27, -40)^T$ .

- в. С помощью разложения и решения системы по пп. а, б найти величину квадратичной формы  $J(x) = x^T Px$ , где  $x$  – решение из п.б.

### Задача 2

Для матрицы

$$P = \begin{pmatrix} 10 & 7 & 3 & 4 \\ 7 & 30 & -6 & 10 \\ 3 & -6 & 9 & 0 \\ 4 & 10 & 0 & 4 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $UU^T$ -разложение матрицы  $P$  ( $U$  – верхняя треугольная матрица с положительными элементами диагонали).  
 б. С помощью  $UU^T$ -разложения матрицы  $P$  решить систему

$$Px = b,$$

с вектором  $b = (4, -7, 0, -2)^T$ .

- в. С помощью разложения и решения системы по пп. а, б найти величину квадратичной формы  $J(x) = x^T Px$ , где  $x$  – решение из п.б.

### Задача 3

Для матрицы

$$P = \begin{pmatrix} 4 & -2 & 2 & 4 \\ -2 & 2 & -3 & 3 \\ 2 & -3 & 14 & -8 \\ 4 & 3 & -8 & 33 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $\overline{L} D \overline{L}^T$ -разложение матрицы  $P$  ( $\overline{L}$  – нижняя треугольная матрица с единицами на диагонали,  $D$  – диагональная матрица с положительными элементами на диагонали).  
 б. С помощью  $\overline{L} D \overline{L}^T$ -разложения матрицы  $P$  решить систему

$$Px = b,$$

с вектором  $b = (8, 0, 5, 32)^T$ .

- в. С помощью разложения и решения системы найти величину квадратичной формы  $J(x) = x^T Px$ , где  $x$  – решение из п.б.

#### Задача 4

Для матрицы

$$P = \begin{pmatrix} 14 & -1 & -1 & -3 \\ -1 & 10 & -2 & 0 \\ -1 & -2 & 5 & 1 \\ -3 & 0 & 1 & 1 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $\overline{U} D \overline{U}^T$ -разложение матрицы  $P$  ( $\overline{U}$  – верхняя треугольная матрица с единицами на диагонали,  $D$  – диагональная матрица с положительными элементами на диагонали).
- б. С помощью  $\overline{U} D \overline{U}^T$ -разложения матрицы  $P$  решить систему

$$Px = b,$$

с вектором  $b = (19, -9, -5, -5)^T$ .

- в. С помощью разложения и решения системы найти величину квадратичной формы  $J(x) = x^T Px$ , где  $x$  – решение из п.б.

## 4.11 Заключение по разделу 4

В данном разделе рассмотрены различные варианты и алгоритмы разложения положительно определённых матриц, известные как разложения Холесского. Значение этого вида разложений велико в области численных методов оптимизации, где широко применяется аппроксимация целевой функции поверхностью второго порядка. Здесь даны теоретические сведения, достаточные для выполнения лабораторного проекта № 2.

В проекте № 2 предлагается выполнить программирование разложения Холесского и на этой основе решить следующие задачи вычислительной линейной алгебры: найти решение СЛАУ для двух типов положительно определённых матриц: заполненных и ленточных (разреженных специальным образом).

Этот проект является вторым базовым для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

# 5

## Проект № 3 «Ортогональные преобразования»

### 5.1 Ортогональные матрицы и приложения

В этом разделе напомним определение и некоторые свойства ортогональных матриц, полезные для дальнейшего.

**Определение 5.1.** Матрица  $T$ , имеющая размер  $(n \times n)$ , т. е.,  $T(n, n)$ , есть *ортогональная матрица*, когда  $TT^T = I$ .

**Свойство А.** Если  $T_1$  и  $T_2$  суть две ортогональные матрицы, то их произведение  $T_1T_2$  есть тоже ортогональная матрица.

**Свойство В.**  $T^{-1} = T^T$  и  $T^T T = I$ .

**Свойство С.** Ортогональное преобразование сохраняет скалярное произведение векторов, т. е.,  $\forall x, y \in \mathbb{R}^n : y^T x \triangleq (x, y) = (Tx, Ty)$ , в частности, оно сохраняет (евклидову) норму вектора:  $\|Ty\| = \|y\|$ .

**Свойство D.** Если  $v$  есть вектор случайных переменных с математическим ожиданием  $\mathbf{E}\{v\} = 0$  и ковариацией  $\mathbf{E}\{vv^T\} = I$ , то теми же характеристиками обладает вектор  $\bar{v} = Tv$ , т. е.,

$$\mathbf{E}\{\bar{v}\} = 0, \quad \mathbf{E}\{\bar{v}\bar{v}^T\} = I.$$

Хотя это свойство легко проверяется, немного удивительно, что компоненты преобразованного вектора остаются взаимно некоррелированы.

Свойства С и D играют существенную роль в квадратно-корневых алгоритмах решения прикладных задач оптимального моделирования и оптимального оценивания методом наименьших квадратов (рис. 5.1).

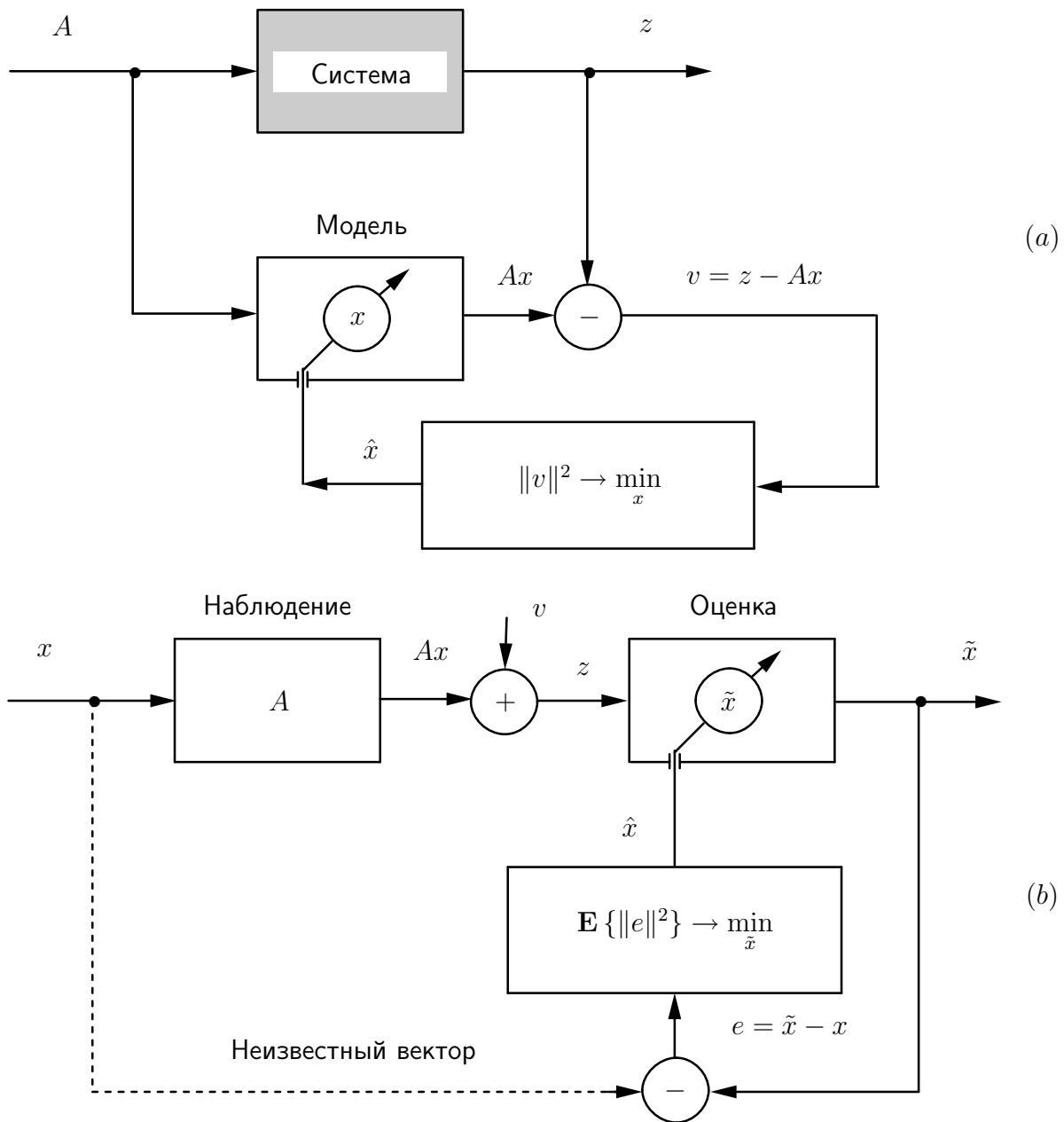


Рис. 5.1. Алгебраически эквивалентные задачи, решаемые методом наименьших квадратов значений невязки  $v$  или среднего квадрата погрешности  $e$ : (a) – оптимальное моделирование неизвестной системы по экспериментальным условиям  $A$  и данным  $z$ ; (b) – оптимальное оценивание неизвестного вектора по наблюдениям  $Ax$  в присутствии случайных помех  $v$  с характеристиками  $E\{v\} = 0$  и  $E\{vv^T\} = I$

## 5.2 Линейная задача наименьших квадратов

Линейная задача наименьших квадратов (см. рис. 5.1) ставится следующим образом (см. также подразд. 11.1) [13, 16].

Дано линейное уравнение

$$z = Ax + v, \quad (5.1)$$

в котором известны вектор  $z \in \mathbb{R}^m$  и  $(m \times n)$ -матрица  $A \in \mathbb{R}^{m \times n}$ , т. е.,  $A = A(m, n)$ . Разностный вектор  $v \triangleq z - Ax$ , называемый *невязкой*, зависит от переменного вектора  $x \in \mathbb{R}^n$ . Требуется найти значение  $\hat{x}$  вектора  $x$ , минимизирующее *квадратический критерий качества*

$$J(x) = (z - Ax)^T(z - Ax) = \|v\|^2 \rightarrow \min. \quad (5.2)$$

Если ни при каком  $x$  невязка  $v$  не может быть обращена в  $\mathbf{0}$  — нулевой вектор, то система  $Ax = z$  — несовместная, в противном случае совместная, т. е., критерий (5.2) охватывает оба случая. Однако сам метод *наименьших квадратов* (МНК), выраженный критерием (5.2), создан *Лежандром* в 1805 году как алгебраическая процедура именно для несовместных систем и подтверждён как статистическая процедура *Гауссом* в 1809 году. МНК как алгебраическая процедура проиллюстрирован выше с помощью рис. 5.1(a), а как статистическая процедура — с помощью рис. 5.1(b). Замечательно, что обе процедуры имеют одинаковые решения, т. е., алгебраически эти решения эквивалентны и при  $\mathbf{E}\{v\} = \mathbf{0}$  и  $\mathbf{E}\{vv^T\} = I$  (см. рис. 5.1(b)) совпадают, поэтому можно говорить о едином МНК-решении  $\hat{x}$ .

МНК-решение  $\hat{x}$  всегда существует как решение *нормальных уравнений*

$$A^T A \hat{x} = A^T z, \quad (5.3)$$

выражается формулой

$$\hat{x} = A^+ z + (I - A^+ A)y \quad (5.4)$$

через произвольный вектор  $y \in \mathbb{R}^n$ , где  $A^+$  — псевдообратная матрица для матрицы  $A$ , и единствено тогда и только тогда, когда  $A^+ A = I$ , что равносильно условию, что только нулевой вектор составляет ядро (нуль-пространство) матрицы  $A$ , т. е., при  $\text{rank } A = n$ .

Условие  $\text{rank } A = n$ , называемое *условием полного столбцового ранга* матрицы  $A$ , обуславливает случай  $m \geq n$ , что при  $m > n$  означает переопределенную систему полного ранга в (5.1). Этот типичный для практики случай ниже и рассматривается, при этом из (5.3), (5.4) следует  $\hat{x} = A^+ z$  и  $A^+ = (A^T A)^{-1} A^T$ .

**Замечание 5.1.** Слагаемое  $\hat{x}_0 \triangleq A^+z$  в (5.4) есть единственное МНК-решение с минимальной нормой, называемое *нормальным псевдорешением*. Оно ортогонально второму слагаемому в (5.4), т. е.,  $A^+z \perp (I - A^+A)y$ , и лежит в пространстве строк матрицы  $A$ , т. е.,  $\hat{x}_0 \in \mathcal{R}(A^T)$ .

Таким образом, типичный для практики случай имеет формальное решение  $\hat{x} = \hat{x}_0 = (A^T A)^{-1} A^T z$ , и вычислительная задача наименьших квадратов заключается в его эффективном отыскании.

### 5.3 Ортогональные матрицы и наименьшие квадраты

В рассматриваемой задаче о наименьших квадратах имеем критерий

$$J(x) = \|z - Ax\|^2, \quad A(m, n), \quad m \geq n, \quad \text{rank } A = n. \quad (5.5)$$

Пусть  $T, T(m, m)$ , есть матрица некоторого ортогонального преобразования. В силу свойства С (см. подразд. 5.1) запишем

$$J(x) = \|T(z - Ax)\|^2 = \|(Tz) - (TA)x\|^2. \quad (5.6)$$

При таком представлении видно, что минимум критерия  $J(x)$ , равный  $J(\hat{x})$ , не зависит от  $T$ . Этим фактом можно воспользоваться, т. е., матрицу  $T$  можно выбрать так, что  $(TA)$  приобретает привлекательную для вычислений форму. Действительно, в подразд. 5.4 и 5.7 мы покажем, как можно выбрать  $T$ , чтобы преобразованная матрица имела вид

$$TA = \begin{bmatrix} R \\ \mathbf{0} \end{bmatrix} \} n \quad (5.7)$$

с верхнетреугольным блоком  $R$ ,  $\text{rank } R = n$ .

Если соответственно этому вектор  $Tz$  разбить на блоки, т. е., записать

$$Tz = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \} n \quad (5.8)$$

то  $J(x)$  от (5.6) приводится к виду

$$J(x) = \|z_1 - Rx\|^2 + \|z_2\|^2. \quad (5.9)$$

Приведение критерия наименьших квадратов к виду (5.9) позволяет заключить, что искомый вектор  $\hat{x}$ , минимизирующий этот критерий, должен удовлетворять уравнению

$$R\hat{x} = z_1, \quad (5.10)$$

которое легко решается *обратной подстановкой* (см. подразд. 5.6), и кроме того,

$$\min J(x) = J(\hat{x}) = \|z_2\|^2. \quad (5.11)$$

В вычислительном отношении эти результаты гораздо более элегантны, чем неразумная траты сил на решение нормальных уравнений (5.3). Однако важнее всего то, что решение, использующее ортогональные преобразования (соотношения (5.7), (5.8), (5.10) и (5.11)), менее чувствительны к погрешностям, вызванным ошибками округления в компьютере. Это видно хотя бы из того, что выражение (5.7) влечёт равенство

$$R^T R = (TA)^T(TA) = A^T A,$$

которое означает, что  $R$  является квадратным корнем из матрицы  $(A^T A)$  системы нормальных уравнений (5.3). Следовательно, при решении системы (5.10) вдвое более эффективно используется разрядная сетка компьютера, чем при решении системы (5.3)<sup>1</sup>.

## 5.4 Преобразование Хаусхолдера

*Преобразования Хаусхолдера* суть матричные представления, которые соответствуют геометрическому понятию отражения [16, 19]. Пусть задан некоторый ненулевой вектор  $u$ , который мы называем *направляющим вектором*. Подпространство, ортогональное ему, есть гиперплоскость  $U_\perp$ . Если взять произвольный вектор  $y$ , то можно отразить его от  $U_\perp$ , в точности соблюдая законы обычного оптического отражения от плоского зеркала (рис. 5.2).

Обозначим отражённый вектор  $y_r$ . Поскольку положение гиперплоскости  $U_\perp$  не зависит от длины направляющего вектора, пронормируем его, т. е., образуем орт  $\hat{u} = u/\|u\|$ . Проекция  $(y \mid u)$  вектора  $y$  на прямую, задаваемую направлением  $u$ , равна  $(y^T \hat{u})\hat{u}$ . Следовательно,

$$y = (y \mid u) + v, \quad v \perp u, \quad v \in U_\perp. \quad (5.12)$$

Отражённый вектор  $y_r$ , как видно из рис. 5.2, имеет разложение

$$y_r = -(y \mid u) + v, \quad v \perp u, \quad v \in U_\perp \quad (5.13)$$

с той же составляющей  $v$ , которая ортогональна вектору  $u$ , но с проекцией

<sup>1</sup> Представление в компьютере квадрата  $a^2$  любого действительного числа  $a$  требует удвоенной разрядности мантиссы, т. е., счет по уравнению (5.10) равносителен счету с удвоенной разрядностью мантиссы чисел по уравнению (5.3).

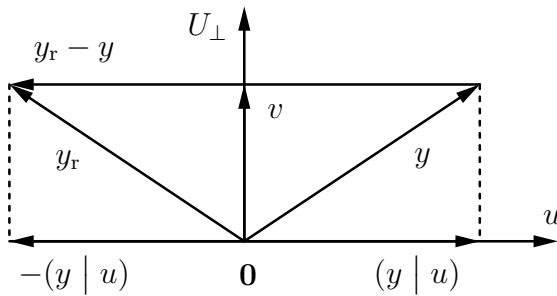


Рис. 5.2. Геометрия преобразования Хаусхолдера. Задача 1 (прямая): даны векторы  $u$  и  $y$ , найти вектор  $y_r$ , отражённый от гиперплоскости  $U_{\perp}$

$-(y \mid u)$ , которая (в силу знака  $-$ ) направлена противоположно проекции  $(y \mid \hat{u})$  вектора  $y$  на направление  $u$ . Исключая  $v$  из (5.12) и (5.13), находим

$$y_r = y - 2(y \mid u)u = (I - \beta uu^T)y = T_u y, \quad (5.14)$$

где  $\beta \triangleq 2/\|u\|^2 = 2/u^T u$ . Матрица Хаусхолдера  $T_u \triangleq (I - \beta uu^T)$ , в вычислениях явно не участвующая, имеет фундаментальное значение для приложений в силу своих замечательных свойств.

**Свойство 1.**  $T_u = T_u^T$ , т. е.,  $T_u$  — симметрическая матрица.

**Свойство 2.**  $T_u^2 = I$ , т. е.,  $T_u$  — идемпотентная матрица. Это легко продемонстрировать алгебраическим разложением матрицы  $T_u^2$  или же геометрически по рис. 5.2 как двукратное отражение вектора  $y$  относительно  $U_{\perp}$ .

**Свойство 3.** Если  $u(j) = 0$ , то  $(T_u y)(j) = y(j)$ , т. е., если  $j$ -я компонента вектора  $u$  — нулевая, то  $T_u$  оставляет  $j$ -ю компоненту вектора  $y$  неизменной.

**Свойство 4.** Если  $u \perp y$ , то  $T_u y = y$ .

**Свойство 5.**

$$T_u y = y - \gamma u, \quad \gamma \triangleq 2y^T u / u^T u = \beta y^T u. \quad (5.15)$$

Свойство 5 — важное с практической точки зрения. Формирование матрицы  $T_u$  в качестве множителя для  $y$  потребовало бы на порядок больше вычислений, чем того требует прямое вычисление  $T_u y$  по свойству 5. Это также означает, что не нужно тратить память для хранения  $T_u$ , что наиболее существенно проявляется при больших  $t$ .

## Триангуляризация матрицы методом Хаусхолдера

Обратимся к основному применению ортогональных преобразований. Для этого решим задачу, обратную к той, что рассмотрена выше, а именно: дан вектор  $y$  и дано желаемое расположение отражённого вектора  $y_r$ , — найти направление  $u$  такое, что  $T_u y = (s, 0, \dots, 0)^T$  (рис. 5.3). Из свойства С, подразд. 5.1, норма (евклидова длина) вектора  $y$  не изменяется при ортогональном преобразовании, следовательно, определим её как

$$\sigma \triangleq \|T_u y\| = |s| = (y^T y)^{1/2}. \quad (5.16)$$

Направление  $u$  может быть получено из свойства 5 (уравнение (5.15)), т. е.,

$$u = \text{const} \cdot (y - s e_1). \quad (5.17)$$

Этот результат приводит к следующему свойству.

**Свойство 6.** Пусть  $s = -\operatorname{sgn}[y(1)]\sigma$ , где  $\operatorname{sgn}[\cdot]$  — функция знака,

$$\operatorname{sgn}[x] = \begin{cases} 1, & x \geq 0, \\ -1, & x < 0, \end{cases}$$

и элементы вектора  $u$  определены выражением (5.17), т. е.,  $u(1) = y(1) - s$ ,  $u(i) = y(i)$ ,  $i > 1$ . Тогда  $T_u y = s e_1$  и  $\beta \triangleq 2/u^T u = -1/(su(1))$ .

**Замечание 5.2.** Геометрический смысл выражения (5.17) ясен из рис. 5.3. Вектор  $y - s e_1$  ортогонален гиперплоскости  $U_\perp$  и коллинеарен вектору  $u$ , а именно:  $y - s e_1 = \gamma u$ . Более того,  $\gamma = 1$ . Выведите это из (5.15).

Непосредственное вычисление  $u^T u$  показывает, что  $u^T u = -2su(1)$ , при этом знак для  $s$  выбран противоположно знаку первого элемента  $y(1)$ , т. е., так, чтобы максимизировать  $|u(1)|$  и тем уменьшить относительную погрешность вычисления разности  $u(1) = y(1) - s$ . Если свойство 6 применить к матрице  $A$ , взяв в качестве  $y$  её первый столбец, то это даст первый шаг, который послужит основой приведения матрицы к верхнетреугольному виду. Повторение таких действий шаг за шагом позволит осуществлять верхнюю триангуляцию любой заданной матрицы  $A$ .

**Лемма 5.1.** Пусть дана матрица  $A(m, n)$ . Тогда существует ортогональное преобразование Хаусхолдера  $T_u$  такое, что

$$T_u A = \underbrace{\begin{matrix} 1 \\ \vdots \\ m-1 \end{matrix}}_{\text{1}} \left\{ \begin{array}{c|c} \overbrace{s}^{1} & \overbrace{\tilde{A}}^{n-1} \\ \hline \mathbf{0} & \tilde{A} \end{array} \right\}. \quad (5.18)$$

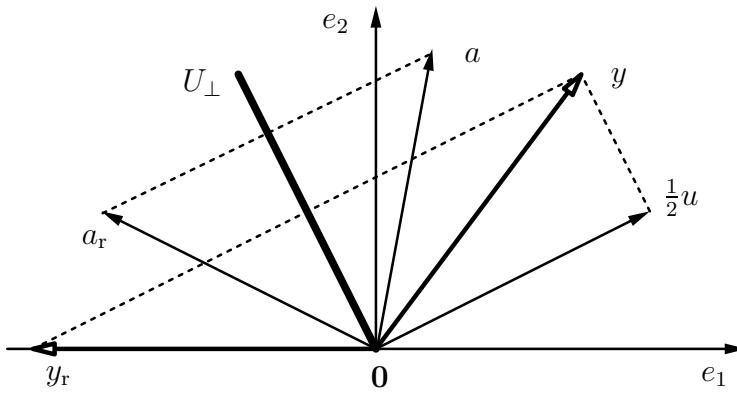


Рис. 5.3. Геометрия преобразования Хаусхолдера. Задача 2 (обратная): даны векторы  $y$  и  $y_r$ , найти вектор  $u$ , задающий отражающую гиперплоскость  $U_\perp$ ; здесь  $y_r = se_1 = [s \mid 0 \cdots 0]^T$ . Докажите, что здесь показан вектор  $\frac{1}{2}u$ , а не  $u$  (см. Замечание 5.2)

**Замечание 5.3.** Скаляр  $s$  и матрица  $\tilde{A}$  в (5.18) вычисляются непосредственно по данным в матрице  $A$ ;  $s$  — по выражению (5.16) и свойству 6, а  $\tilde{A}$  — по свойству 5, (5.15). Первый столбец, который уместно назвать *ведущим столбцом в преобразовании Хаусхолдера*, используют как вектор  $y$  в задаче 2 (см. рис. 5.3) для определения вектора  $u$ . Второй и далее столбцы, обозначенные на рис. 5.3 произвольно как вектор  $a$ , отражают от найденной таким образом гиперплоскости  $U_\perp$ , решая для этого задачу 1 (см. рис. 5.2) с  $y := a$  и тем самым получая блок  $\tilde{A}$ .

**Теорема 5.1 (Триангуляризация матрицы по методу Хаусхолдера).** Пусть  $A_1 := A(m, n)$  и для каждого  $j$  выбрано элементарное преобразование Хаусхолдера  $T_j$  так, что

$$T_j A_j = \begin{cases} \begin{array}{c|c} 1 & \\ \hline s_j & a_j^T \\ \mathbf{0} & A_{j+1} \end{array} \end{cases}, \quad j = 1, \dots, k; \quad k \leq \min(m-1, n). \quad (5.19)$$

Тогда в процессе после  $k$  повторных применений свойства 6 и леммы 5.1

имеем следующий промежуточный результат триангуляризации матрицы  $A$ :

$$T^{(k)} A = \begin{bmatrix} & s_1 & & & a_1^T \\ & & s_2 & & a_2^T \\ & & & \ddots & \vdots \\ & & & & s_k & a_k^T \\ 0 & & & & & A_{k+1} \end{bmatrix} \quad (5.20)$$

с отвечающей этому моменту процесса итоговой матрицей преобразований

$$T^{(k)} = \begin{bmatrix} I_{k-1} & 0 \\ 0 & T_k \end{bmatrix} \cdots \begin{bmatrix} I_1 & 0 \\ 0 & T_2 \end{bmatrix} T_1. \quad (5.21)$$

**Замечание 5.4.** Важно подчеркнуть, что алгоритм триангуляризации (5.19) не требует вычисления или запоминания ортогональной матрицы  $T^{(k)}$ , так как правая часть равенства (5.4) вычисляется непосредственно в соответствии с замечанием 5.3. Стоит также заметить, как неявно определяется  $A_{j+1}$  рекурсией по  $j$  в алгоритме (5.19). Кроме  $A_{j+1}$ , на шаге  $j$  этой рекурсии определяются скаляр  $s_j$  и  $(n - j)$  компонент вектор-строки  $a_j^T$ . Эти неявные соотношения для  $s_j$ ,  $a_j^T$  и  $A_{j+1}$  и весь процесс вычислений (рис. 5.4) представлены явном виде в подразд. 5.5.

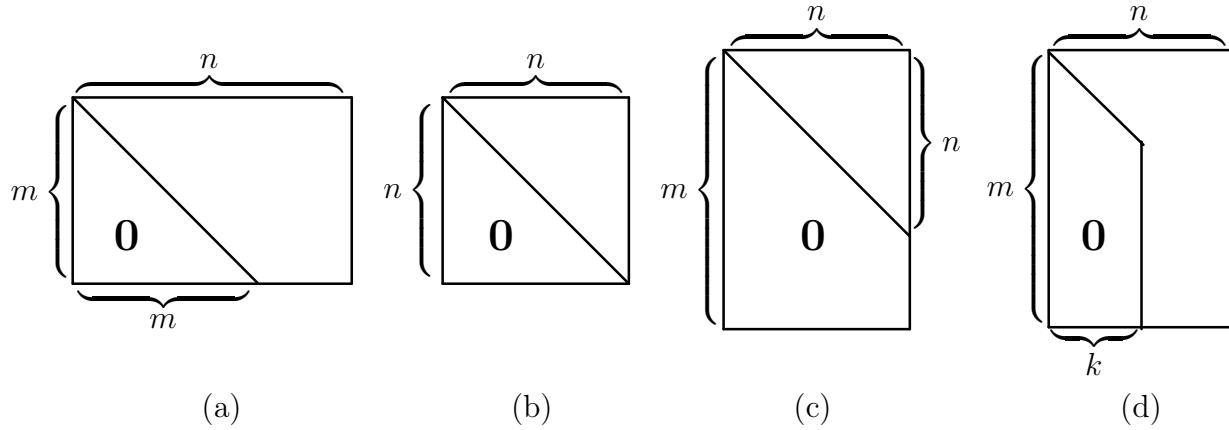


Рис. 5.4. Представление возможных случаев применения теоремы 5.1 к матрице  $A(m, n)$ :  
(a) недоопределённая система:  $k = m - 1 \leq n$ ; (b) определённая система:  $k = n - 1$ ,  $m = n$ ; (c) переопределённая система:  $k = n < m$ ; (d)  $k < n < m$

## 5.5 Шаг триангуляризации матрицы методом Хаусхолдера

Пусть матрица  $A = A(m, n)$  задана. Тогда, согласно лемме 5.1, базовая операция процесса триангуляризации заключается в вычислении скаляра  $s$  и матрицы  $\tilde{A} = \tilde{A}(m, n - 1)$  таких, что

$$T_u A = \begin{cases} \begin{array}{c|c} 1 & n-1 \\ \hline s & \tilde{A} \\ \mathbf{0} & \end{array} \end{cases}_{m-1} \quad (5.22)$$

**Алгоритм.** Для вычисления  $s$  следует применять свойство 6 (см. стр. 172), т. е., выполнять (5.23) для всех  $k = 1$  до  $\min(m - 1, n)$ . Затем для вычисления  $\tilde{A}$  следует применять свойство 5 (см. стр. 171), т. е., последовательно в цикле по  $j = 2, \dots, n$  для всех  $i = k, \dots, m$  выполнять (5.24). Здесь  $\lambda \triangleq -\gamma$  (см. (5.15)),  $\alpha \triangleq -\beta$  (см. (5.14)) и использовано свойство 6).

Для  $k = 1$  до  $\min(m - 1, n)$

$$\left. \begin{array}{l} s_k = -\operatorname{sgn}[A(k, k)] \left( \sum_{i=k}^m [A(i, k)]^2 \right)^{1/2}, \\ u_k(1) = A(k, k) - s_k, \\ u_k(i) = A(k + i - 1, k), \quad i = 2, \dots, m - k + 1, \\ \alpha_k = 1/(s_k u_k(1)) \quad (\alpha_k < 0). \end{array} \right\} \quad (5.23)$$

Для  $j = k + 1, \dots, n$

$$\left. \begin{array}{l} \lambda := \alpha_k \cdot \sum_{i=k}^m u_k(i - k + 1) A(i, j), \\ A(i, j) := A(i, j) + \lambda u_k(i - k + 1). \end{array} \right\} \quad (5.24)$$

Приведённый алгоритм (5.23), (5.24) называют *столбцово ориентированным алгоритмом Хаусхолдера*, так как операции (5.24) вычисляют целиком каждый  $j$ -й столбец матрицы, находящийся справа от ведущего, т. е.,  $k$ -го столбца. Альтернативная схема вычислений называется *строчно ориентированным алгоритмом Хаусхолдера*. Её можно получить из выражения  $T_u = I - \beta u u^T$  для матрицы Хаусхолдера следующим образом.

Введём вспомогательные обозначения:  $\mu \triangleq \sqrt{\beta}$ ,  $w \triangleq \mu u$ , чтобы записать  $T_u = I - ww^T$ . Тогда  $(T_u A) = A - wz^T$ , где  $z^T \triangleq w^T A = \mu v^T$ ,  $v^T \triangleq \sum_{i=1}^m u(i)A(i, \cdot)$  и  $A(i, \cdot)$  есть  $i$ -я строка матрицы  $A = A(m, n)$ . Введём обозначение  $\lambda^T = \alpha v^T$ , используя ранее введённое (см. (5.23))  $\alpha \triangleq -\beta$ . Отсюда получаем формулу для любой  $i$ -й строки  $(T_u A)(i, \cdot)$  преобразованной матрицы  $(T_u A)$  в виде

$$(T_u A)(i, \cdot) = A(i, \cdot) - w(i)z^T = A(i, \cdot) - \mu^2 u(i)v^T = A(i, \cdot) + \lambda^T u(i).$$

**Алгоритм** (строчно ориентированный), эквивалентный (5.23) и (5.24).

Для $k = 1$ до $\min(m - 1, n)$	$s_k = -\operatorname{sgn}[A(k, k)] \left( \sum_{i=k}^m [A(i, k)]^2 \right)^{1/2},$ $u_k(1) = A(k, k) - s_k,$ $u_k(i) = A(k + i - 1, k), \quad i = 2, \dots, m - k + 1,$ $\alpha_k = 1/(s_k u_k(1)) \quad (\alpha_k < 0).$
---------------------------------	--

(5.25)

Для $j = k + 1, \dots, n$	$\lambda_k(j - k) := \alpha_k \cdot \sum_{i=k}^m u_k(i - k + 1) A(i, j),$
---------------------------	---

Для  $i = k, k + 1, \dots, m$

Для $j = k + 1, \dots, n$	$A(i, j) := A(i, j) + \lambda_k(j - k) u_k(i - k + 1).$
---------------------------	---

(5.26)

## 5.6 Решение треугольной системы и обращение матриц

Как отмечено в подразд. 5.3, мы часто заинтересованы в решении уравнения

$$Rx = z, \quad (5.27)$$

где  $R = R(n, n)$  – верхняя треугольная невырожденная матрица. Если нужно иметь только решение  $x$ , то  $R^{-1}$  (для  $x = R^{-1}z$ ) вычислять не надо. Следующий алгоритм обратной подстановки позволяет вычислить решение  $x$  непосредственно.

**Алгоритм.** Для  $j = n, n - 1, \dots, 1$  вычислять

$$x(j) = \left( z(j) - \sum_{k=j+1}^n R(j, k)x(k) \right) / R(j, j). \quad (5.28)$$

По сложности этот алгоритм почти такой же, как матричное умножение. Он допускает записывать  $x(j)$  поверх  $z(j)$ , что очень удобно в приложениях.

Если всё же требуется иметь матрицу  $U \triangleq R^{-1}$ , то её можно вычислять по алгоритму окаймления, основанному на следующем легко проверяемом тождестве для треугольных матриц:

$$\begin{bmatrix} R_j & y \\ 0 & \sigma_{j+1} \end{bmatrix}^{-1} = \begin{bmatrix} R_j^{-1} & -R_j^{-1}y\sigma_{j+1}^{-1} \\ 0 & \sigma_{j+1}^{-1} \end{bmatrix} = R_{j+1}^{-1}. \quad (5.29)$$

Это соотношение позволяет вычислять матрицу  $U \triangleq R^{-1}$  рекуррентно, а именно: если  $R_j^{-1} = U_j$ , где  $R_j$  обозначает верхнюю левую часть матрицы  $R$ , то

$$U_{j+1} = \begin{bmatrix} U_j & -U_j [R(1, j+1), \dots, R(j, j+1)]^T \sigma_{j+1} \\ 0 & \sigma_{j+1} \end{bmatrix}, \quad (5.30)$$

где  $\sigma_{j+1} = 1/R(j+1, j+1)$ . Полагая  $U = R_{-1}$ , этот результат представим в алгоритмической форме.

**Алгоритм.** Обращение верхней треугольной матрицы. Задать начальное значение

$$U(1, 1) = 1/R(1, 1). \quad (5.31)$$

Для  $j = 2, \dots, n$  вычислять по формулам (5.32) и (5.33):

$$U(j, j) = 1/R(j, j), \quad (5.32)$$

$$U(k, j) = - \left( \sum_{i=k}^{j-1} U(k, i)R(i, j) \right) U(j, j), \quad k = 1, \dots, j-1. \quad (5.33)$$

**Замечание 5.5.**  $R^{-1}$  вычисляется по столбцам, при этом элементы матрицы  $R^{-1}$  могут записываться поверх элементов исходной матрицы  $R$ .

В справочных целях приведём примеры реализации данного алгоритма на языке FORTRAN. Эти примеры могут помочь студентам написать свои собственные программы на других языках высокого уровня при выполнении лабораторного проекта № 3, описание которого дано ниже в подразд. 5.16.

**Обращение верхней треугольной матрицы:**  $U := R^{-1}$ . Реализуются формулы (5.31), (5.32) и (5.33). Если нужно,  $U$  может замещать  $R$  [16].

---

$R(N, N)$ ,  $U(N, N)$ ,  $R$  и  $U$  – верхние треугольные матрицы

---

```
U(1, 1) = 1./R(1, 1)
DO 20 J = 2, N
  U(J, J) = 1./R(J, J)
  JM1 = J - 1
  DO 20 K = 1, JM1
    SUM = 0.
    DO 10 I = K, JM1
      10 SUM = SUM - U(K, I) * R(I, J)
      20 U(K, J) = SUM * U(J, J)
```

---

В случаях, когда важно или нужно экономить память компьютера, матрицы в программе объявляют как одномерные массивы (см. подразд. 4.3). Хотя в компьютере даже многомерно объявленные массивы всегда хранятся как одномерные, компилятор генерирует индексные выражения с операциями умножения и деления. Операции сложения и вычитания в компьютерах выполняются гораздо быстрее, поэтому индексы для доступа к элементам матриц следует программировать в рекуррентной инкрементной форме, экономя таким образом время процессора (табл. 5.1). В этой программе преобразование в треугольную форму выполняется отождествлением  $J(J-1)/1+I$  с  $(I, J)$ . Рекуррентное инкрементное вычисление  $KK$ ,  $JJ$  и  $KK$  экономит вычисления.

Как отмечалось на с. 177, иногда требуется вычислять  $R^{-1}$ . Такая ситуация появляется, если требуется найти  $A^{-1}$ , для которой уже выполнено преобразование  $TA = R$ , где  $T = T^{(n-1)}$  по формуле (5.21), так как в теореме 5.1 для этого случая  $m = n$  и  $A^{-1} = R^{-1}T$ . Последнее означает, что то же самое ортогональное преобразование  $T$  теперь надо применить к строкам матрицы  $R^{-1}$ , но уже в обратном порядке следования элементарных преобразований, составляющих полное преобразование  $T = T^{(n-1)}$  по формуле (5.21). Таким образом, возникает проблема запоминания элементарных преобразований, составляющих полное преобразование  $T = T^{(n-1)}$ , чтобы позже можно было его применить в задаче отыскания  $A^{-1}$  или же для решения уравнения  $Ax = z$  с невырожденной матрицей  $A$  после преобразования  $TA = R$ .

Таблица 5.1. Эффективное обращение верхней треугольной матрицы  $U := R^{-1}$ 

$R(N, N), \quad U(N, N), \quad R \text{ и } U - \text{верхнетреугольные матрицы}$	
$U(1) = 1./R(1)$	© $R(1) \equiv R(1, 1)$
$JJ = 1$	
DO 20 $J = 2, N$	
$\overline{JJ} = JJ$	© $\overline{JJ} = J(J - 1)/2 = (J - 1, J - 1)$
$JJ = JJ + J$	© $JJ = J(J + 1)/2 = (J, J)$
$U(JJ) = 1./R(JJ)$	
$JM1 = J - 1$	
$KK = 0$	
DO 20 $K = 1, JM1$	
$KK = KK + K$	© $KK = K(K + 1)/2$
$KI = KK$	
$SUM = 0.$	
DO 10 $I = K, JM1$	
$SUM = SUM - U(KI) * R(\overline{JJ} + I)$	© $KI = (K, I), \overline{JJ} + 1 = (I, J)$
10 $KI = KI + I$	© $KI = (K, I + 1)$
20 $U(\overline{JJ} + K) = SUM * U(JJ)$	© $\overline{JJ} + K = (K, J)$

Реализуются формулы (5.31), (5.32) и (5.33). Верхние треугольные матрицы  $R$  и  $U$  хранятся как векторы размерности  $N(N + 1)/2$ . Если нужно,  $U$  может замещать  $R$ .

Как видно из (5.24), для отражения вектора  $y = T^{(k)}z$  от гиперплоскости  $U_{\perp}$ , заданной вектором  $u$ , требуется иметь сохранёнными две величины: вектор  $u$  и скаляр  $\alpha$ . Поскольку нули ниже диагонали, получающиеся в результате триангуляризации, хранить не нужно, это место можно отвести для сохранения вектора  $u$  (вместе с диагональю, поэтому диагональные элементы  $s_k$  приходится хранить отдельно). Данное предложение схематически показано на рис. 5.5 (вверху). Каким образом можно выполнить вычисление матрицы  $A^{-1}$ , показано на рис. 5.5 (внизу) на конкретном примере размерностей, а именно:  $m = 4$ ,  $n = 4$ .

## 5.7 Преобразование Гивенса

Преобразование Гивенса осуществляет вращение вектора в плоскости двух координат. Очевидно, поворот вектора  $y = (y_1 \mid y_2)^T$  на угол  $\theta$  по часовой стрелке эквивалентен повороту системы координат против часовой стрелки на тот же угол.

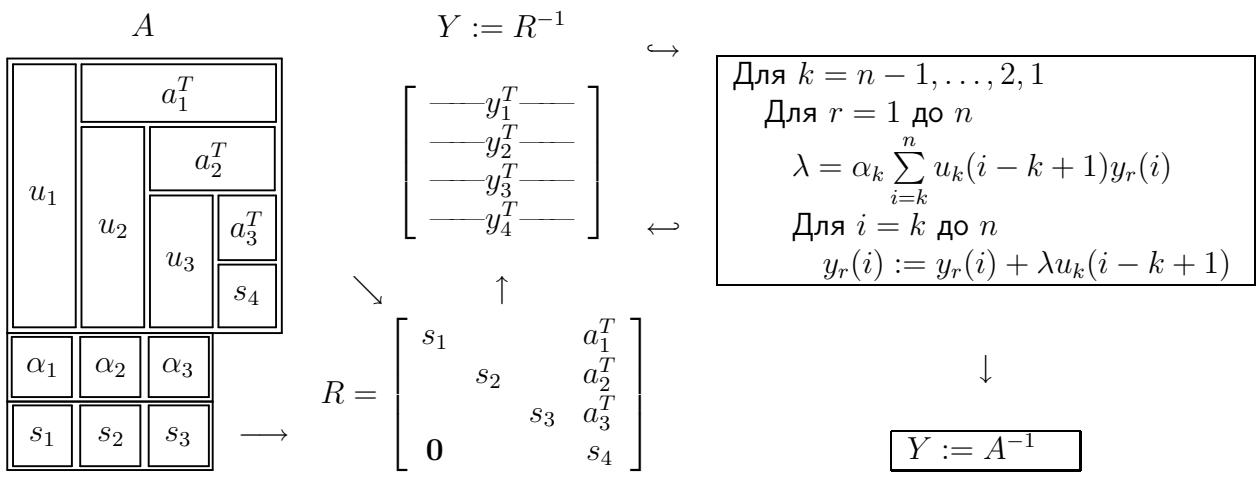
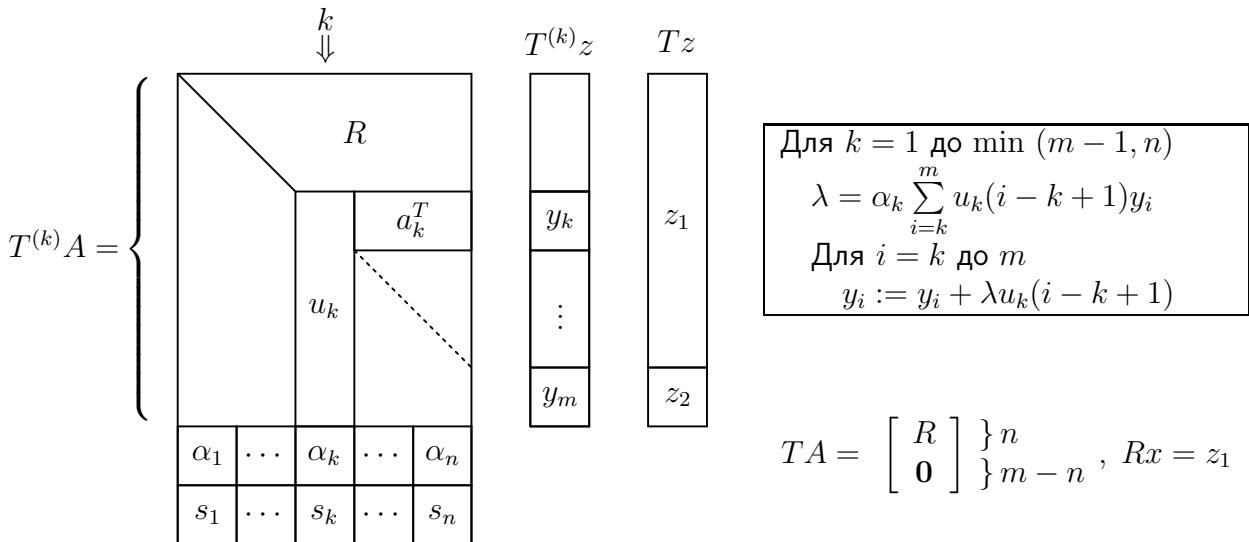


Рис. 5.5. Вверху: Сохранение преобразования  $T$  и вычисление вектора  $y = Tz, \forall y \in \mathbb{R}^m$ .  
Внизу: Вычисление матрицы  $A^{-1}$  после сохранения преобразования  $T$

Следовательно, легко найти (рис. 5.6), что координаты  $y'_1, y'_2$  повёрнутого вектора  $y_r = (y'_1 \mid y'_2)^T$  определяются в виде  $y'_1 = y_1 \cos \theta + y_2 \sin \theta$ ,  $y'_2 = -y_1 \sin \theta + y_2 \cos \theta$ .

Записывая это в матричной форме и требуя, чтобы поворот  $P_{1,2}$  в плоскости  $(e_1, e_2)$  происходил до совмещения с первой координатной осью, получим

$$y_r = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} y = P_{1,2} y = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad \left\{ \begin{array}{l} c \triangleq \cos \theta = y_1/r \\ s \triangleq \sin \theta = y_2/r \end{array} \right\}, \quad r \triangleq \sqrt{y_1^2 + y_2^2},$$

где, очевидно, матрица  $P_{1,2}$  плоского вращения ортогональна при любом  $\theta$ .

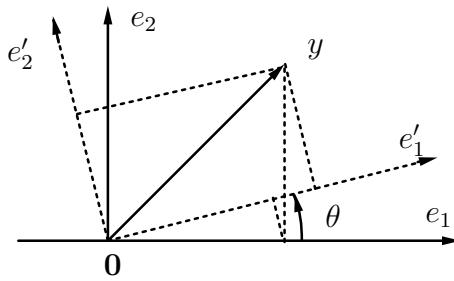


Рис. 5.6. Геометрия вращений

### Триангуляризация матрицы преобразованиями Гивенса

Выбор  $\theta$  такой, что вторая координата вектора  $y_r$  становится равной нулю, используем для триангуляризации матрицы  $A(m, n)$ . На первом шаге нужны преобразования, делающие равными нулю все элементы ниже первого диагонального элемента. Для этого, очевидно, нужно выполнить последовательно элементарные вращения  $P_{1,2}, P_{1,3}, \dots, P_{1,m}$ . Так определённые преобразования воздействуют на все столбцы матрицы, но только первый столбец, который уместно назвать *ведущим столбцом в преобразовании Гивенса*, приобретает желаемый вид.

**Лемма 5.2.** Пусть дана матрица  $A(m, n)$  и  $y$  – её ведущий (левый) столбец. Тогда существует ортогональное преобразование Гивенса, задаваемое матрицей  $P_1 = P_1(m, m)$ , такое, что

$$P_1 A = \begin{cases} \begin{array}{c|c} 1 & n-1 \\ \hline r & \tilde{A} \\ \hline 0 & \end{array} \end{cases}, \quad (5.34)$$

$$P_1 = P_{1,m} \cdots P_{1,3} P_{1,2},$$

и матрицы  $P_{1,j}$  определяются по алгоритму на рис. 5.7.

**Теорема 5.2 (Триангуляризация матрицы по методу Гивенса).** Пусть  $A_1 := A(m, n)$  и для каждого  $j = 1, 2, \dots, k$ ,  $k \leq \min(m - 1, n)$  серия элементарных преобразований Гивенса, задаваемая матрицей  $P_j$  размера  $(m + 1 - j) \times (m + 1 - j)$ , выбрана, как сказано ниже.

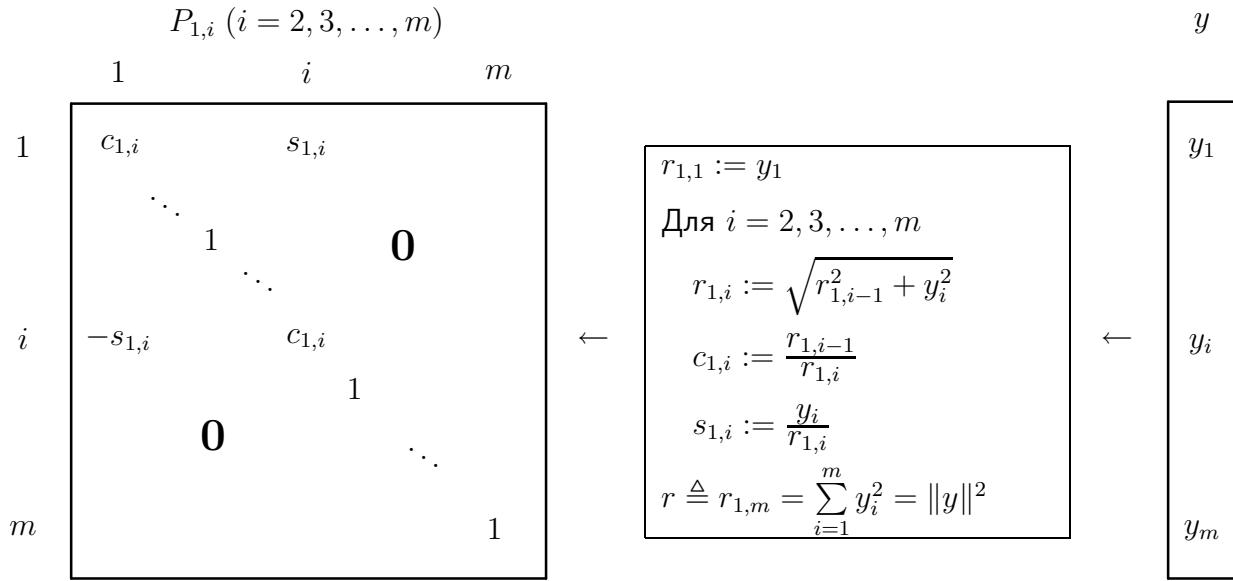


Рис. 5.7. Вычисление матрицы  $P_{1,j}$

Для ведущего (левого) столбца  $y_j$  матрицы  $A_j$  эта  $P_j$  выбрана так, что

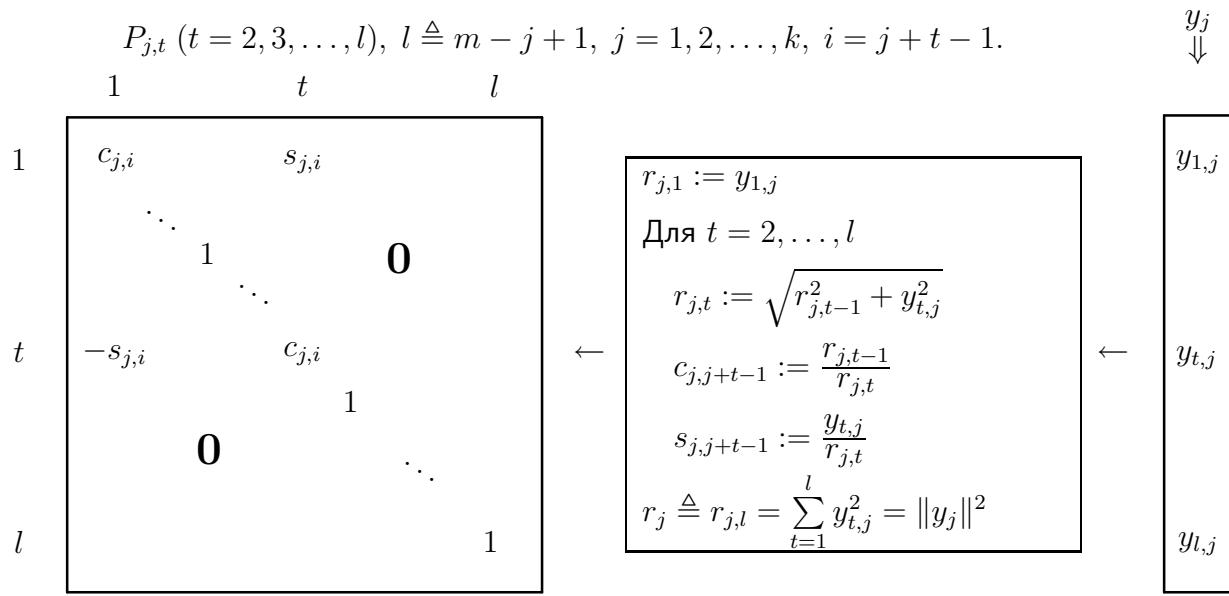
$$P_j A_j = \begin{cases} \begin{array}{c|c} 1 & n-j \\ \hline r_j & a_j^T \\ \mathbf{0} & A_{j+1} \end{array} \end{cases}, \quad j = 1, \dots, k; \quad k \leq \min(m-1, n). \quad (5.35)$$

Тогда после  $k$  повторных применений леммы 5.2 имеем следующий промежуточный результат триангуляризации матрицы  $A$ :

$$P^{(k)} A = \left[ \begin{array}{ccccc} r_1 & & & & a_1^T \\ & r_2 & & & a_2^T \\ & & \ddots & & \vdots \\ & & & r_k & a_k^T \\ \mathbf{0} & & & & A_{k+1} \end{array} \right] \quad (5.36)$$

с отвечающей этому моменту процесса итоговой матрицей преобразований

$$P^{(k)} = \begin{bmatrix} I_{k-1} & 0 \\ 0 & P_k \end{bmatrix} \cdots \begin{bmatrix} I_1 & 0 \\ 0 & P_2 \end{bmatrix} P_1, \quad P_j = P_{j,m-j+1} \cdots P_{j,3} P_{j,2}, \quad (5.37)$$



Формула (5.37) имеет рекуррентный вид произведения

$$\left. \begin{aligned} P^{(j)} &= \left[ \begin{array}{cc} I_{j-1} & 0 \\ 0 & P_j \end{array} \right] P^{(j-1)}, \quad P^{(1)} = P_1 \\ P_j &= P_{j,m-j+1} \cdots P_{j,3} P_{j,2}, \quad j = 2, \dots, N \end{aligned} \right\}, \quad N = \min(m-1, n). \quad (5.38)$$

Все участвующие здесь матрицы являются ортогональными, поэтому финальная матрица  $P \triangleq P^{(N)}$  также ортогональна. Общее число используемых при этом элементарных матриц вращения равно  $(m-1) + (m-2) + \dots + (m-N) = (2m-N-1)N/2$ . В результате (в случае  $m > n$ ) получим

$$PA = \begin{bmatrix} R \\ \vdots \\ 0 \end{bmatrix}, \quad R = \boxed{\begin{array}{c|c} & \diagdown \\ & \diagdown \\ \diagup & \diagup \\ 0 & \end{array}} \triangleq R_{ne}, \quad (5.39)$$

где индекс  $ne$  подчёркивает, что в треугольной матрице  $R$  заполненной частью может быть только «северо-восточная» (*north-by-east*) часть. Полагая  $Q = P^T$ , при  $m = n$  имеем  $QR$ -разложение матрицы  $A = A(n, n)$ , т. е.,  $A = QR$ . Матрицы в (5.37) и (5.38) непосредственно не вычисляются.

Для алгоритма Гивенса – так же, как и для других матричных алгоритмов, – имеются две схемы вычислений: (1) строчно ориентированная схема Гивенса и (2) столбцово ориентированная схема Гивенса (рис. 5.8). Как и в алгоритме преобразований Хаусхолдера (см. рис. 5.5), здесь обычно требуется сохранять информацию о произведённых по ходу алгоритма (5.38) элементарных преобразованиях.

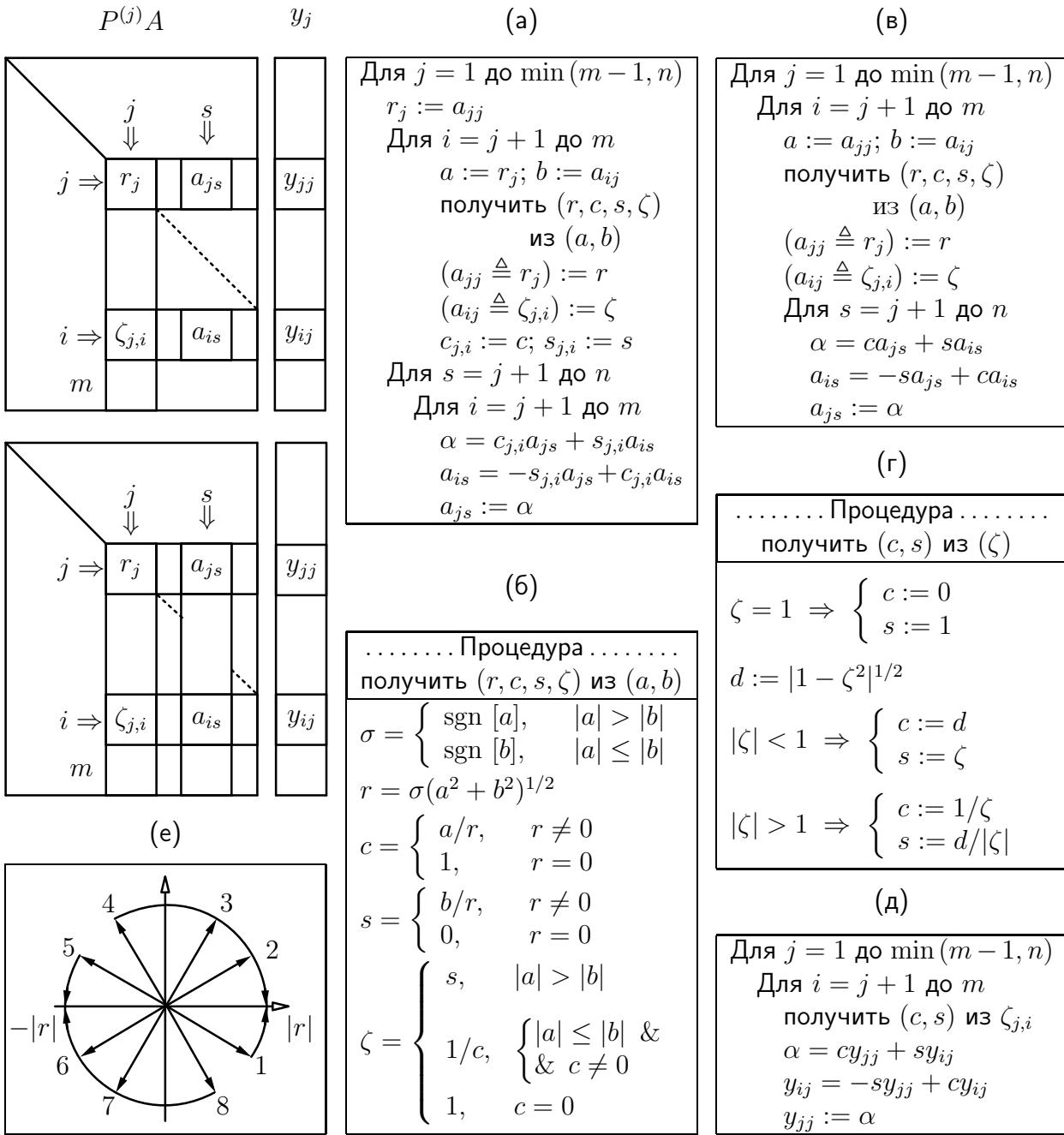


Рис. 5.8. Преобразование Гивенса: (а) столбцово ориентированная схема вычисления матрицы  $PA$ , где  $P = P^{(j)}$  при  $j = \min(m - 1, n)$  (нижняя матрица слева); (б) вычисление координаты  $r$  вектора  $(a, b)^T$ , повернутого до совмещения с первой осью, а также косинуса и синуса угла поворота и рабочего признака  $\zeta$ ; (в) строчно ориентированная схема вычисления матрицы  $PA$  (верхняя матрица слева); (г) восстановление косинуса и синуса угла поворота из признака  $\zeta$ ; (д) получение вектора  $u$  теми преобразованиями  $P_{j,i}$  произвольного вектора  $z \in \mathbb{R}^m$ , которые сохранены в рабочих признаках  $\zeta_{j,i}$  и восстанавливаются из них; (е) вследствие п. (б) векторы 1, 2, 3 и 4 поворачиваются к положительному направлению первой координатной оси, а векторы 5, 6, 7 и 8 – к отрицательному направлению этой оси

Сохранение этой информации позволит впоследствии решать системы уравнений  $Ax = z$  (совместные или несовместные, в последнем случае — по методу наименьших квадратов, см. подразд. 5.3) или же находить обратную матрицу  $A^{-1}$  (когда  $m = n$ ).

Необходимая информация — это значения косинуса и синуса, но их сохранение было бы неэффективным решением. Gentleman (1973) предложил способ [12], включённый в рис. 5.8(б) и (г) с геометрической иллюстрацией его действия на рис. 5.8(е). Введённый им рабочий признак  $\zeta$  — это одно число, которое можно хранить в позиции  $(i, j)$  как  $\zeta_{j,i}$  вместо нулевого элемента, появляющегося в позиции  $(i, j)$  матрицы (5.39) в момент преобразования  $P_{j,t}$  ( $t = i+1-j$ ) в (5.37). Как и с преобразованиями Хаусхолдера, нахождение  $A^{-1}$  после преобразований Гивенса требует такой же последовательности процедур: сначала находят  $R^{-1}$  (см. рис. 5.5 (внизу)), затем к  $R^{-1}$  применяют с правой стороны финальное преобразование  $P \triangleq P^{(N)}$  (5.38), так как  $A^{-1} = R^{-1}P$ . Для этого для рис. 5.5 (внизу) надо взять алгоритм из рис. 5.8(д), который также отыскивает  $Pz$  при решении уравнения  $Ax = z$ .

## 5.8 Варианты заполнения матрицы $R$

Традиционно ортогональные преобразования (выше рассмотрены  $T$  — преобразование Хаусхолдера и  $P$  — преобразование Гивенса) приводят матрицу к виду, показанному на рис. 5.4 или в выражении (5.39). Однако выбор того угла матрицы, который должен остаться треугольно заполненным, естественно, произволен. Предпочтения диктуются целями использования, т. е., предназначением преобразования. Преследуя здесь учебно-тренировочные цели, включим в проект (см. подразд. 5.16) все четыре возможных варианта заполнения матрицы  $R$ . Вариант 1 показан в (5.39), другие три имеют следующий вид. Вариант 2:

$$QA = \begin{bmatrix} R \\ \cdots \\ 0 \end{bmatrix}, \quad R = \begin{array}{|c|c|} \hline & \diagdown \\ \diagup & 0 \\ \hline \end{array} \triangleq R_{\text{nw}}, \quad (5.40)$$

где  $Q$  обозначает либо  $T$  (преобразование Хаусхолдера), либо  $P$  (преобразование Гивенса), индекс <sub>nw</sub> подчёркивает, что в треугольной матрице  $R$  заполненной частью может быть только «северо-западная» (*north-by-west*) часть.

Вариант 3:

$$QA = \begin{bmatrix} 0 \\ \cdots \\ R \end{bmatrix}, \quad R = \begin{array}{c|c} & 0 \\ \hline 0 & \end{array} \triangleq R_{se}, \quad (5.41)$$

где индекс  $se$  подчёркивает, что в треугольной матрице  $R$  заполненной частью может быть только «юго-восточная» (*south-by-east*) часть. Вариант 4:

$$PA = \begin{bmatrix} 0 \\ \cdots \\ R \end{bmatrix}, \quad R = \begin{array}{c|c} 0 & \\ \hline & \end{array} \triangleq R_{sw}, \quad (5.42)$$

где индекс  $sw$  подчёркивает, что в треугольной матрице  $R$  заполненной частью может быть только «юго-западная» (*south-by-west*) часть. Вполне очевидно, что эти варианты получаются простым изменением порядка действий в алгоритмах преобразований.

## 5.9 Правосторонние ортогональные преобразования

С правосторонними ортогональными преобразованиями мы уже сталкивались (см. подразд. 5.6); тогда для квадратной матрицы  $A$  после  $TA = R$  вычисляли  $A^{-1} = R^{-1}T$ . Однако можно начинать с правостороннего преобразования матрицы  $A$ ; тогда отыскание  $A^{-1}$  потребует, соответственно, левостороннего преобразования.

Пусть  $A = A(n, n)$  – квадратная невырожденная матрица. Будем рассматривать её строки как векторы в  $\mathbb{R}^n$ . Преобразования вектора как матрицы-строки в  $n$ -мерном линейном пространстве задаются умножением её на преобразующую матрицу справа. Поэтому правосторонним ортогональным преобразованием  $Q$  можно привести матрицу  $A$  к виду  $AQ = R$ , где применена ортогональная матрица  $Q$  одного из типов, а  $R$  – треугольная матрица, имеющая форму одного из возможных вариантов заполнения (см. подразд. 5.8). При этом преобразованию  $Q$  подвергаются не столбцы, а строки матрицы  $A$ , и преобразование  $Q$  запоминается по принципу, показанному ранее на рис. 5.5 и рис. 5.8, на месте элементов, обращаемых в нуль.

После такого преобразования матрицы  $A$  решение системы  $Ax = z$  сводится к решению эквивалентной системы с треугольной матрицей  $Ry = z$ . Затем искомый вектор определяется через сохранённое преобразование  $Q$  как  $x = Qy$ . Обратная матрица  $A^{-1}$ , соответственно, находится как решение системы

$RY = I$  с последующим преобразованием  $Q$  матрицы  $Y$ , т.е.,  $X = A^{-1} = QY$ . Матрица  $Q$  не формируется, из чего видна необходимость запоминания преобразований, обеспечивших  $AQ = R$ .

## 5.10 Двусторонние ортогональные преобразования

Ортогональные преобразования, будучи применены одновременно слева и справа к данной матрице  $A$ , позволяют приводить её к формам с нулями как ниже, так и выше диагонали. Это, в свою очередь, облегчает решение других сложных задач (матричная проблема собственных значений [30]). С помощью ортогональных преобразований для квадратной матрицы широко распространены: приведение симметрической матрицы к трёхдиагональному виду и приведение квадратной матрицы к двухдиагональному виду. При этом в качестве ортогональных преобразований одинаково успешно могут быть использованы преобразования Хаусхолдера или преобразования Гивенса.

### Приведение симметрической матрицы к трёхдиагональному виду

Применим к симметрической матрице слева и справа преобразование Хаусхолдера (или Гивенса), выбирая его из задачи желаемого преобразования ведущего столбца и ведущей строки, а именно: сохранение первого диагонального элемента, получение ненулевых элементов в двух смежных с ним позициях и получение нулевых элементов в остальных позициях.

**Лемма 5.3.** Пусть дана матрица  $A = A(n, n) = A^T$ . Тогда существует ортогональное преобразование  $Q_2$  (Хаусхолдера  $T_2$  или Гивенса  $P_2$ ) такое, что

$$\left[ \begin{array}{c|c} I_1 & \mathbf{0} \\ \mathbf{0} & Q_2 \end{array} \right] A \left[ \begin{array}{c|c} I_1 & \mathbf{0} \\ \mathbf{0} & Q_2^T \end{array} \right] = \underbrace{\left[ \begin{array}{c|cc} a_1 & s_1 & \mathbf{0} \\ s_1 & \hline \mathbf{0} & \tilde{A} \end{array} \right]}_{\substack{1 \\ n-2}}. \quad (5.43)$$

**Замечание 5.6.** В (5.43) транспонирование  $Q_2^T$  не требуется, если в качестве  $Q_2$  взято преобразование Хаусхолдера (в силу его симметричности). При этом индекс « $_2$ » указывает на позицию того элемента в ведущем столбце (для левостороннего преобразования) или в ведущей строке (для правостороннего преобразования), который остаётся ненулевым в этом столбце (в результате

применения  $Q_2$ ) или в этой строке (в результате применения  $Q_2^T$ ). В данном случае, т. е., в (5.43), эти элементы суть  $s_1$  и  $s_1$ . Элемент  $a_1$  не изменяется, так как  $I_1$  – единичная матрица размера  $1 \times 1$ .

**Теорема 5.3** (Тридиагонализация симметрической матрицы). Пусть дана симметрическая матрица  $A = A(n, n) = A^T$ ,  $A_1 := A(n, n)$  и для каждого  $j = 1, \dots, k$ , где  $k \leq N = n - 2$ , выбрано элементарное преобразование  $Q_{j+1}$  (Хаусхолдера  $T_{j+1}$  или Гивенса  $P_{j+1}$ ) так, что

$$\left[ \begin{array}{c|c} I_1 & \mathbf{0} \\ \mathbf{0} & Q_{j+1} \end{array} \right] A_j \left[ \begin{array}{c|c} I_1 & \mathbf{0} \\ \mathbf{0} & Q_{j+1}^T \end{array} \right] = \underbrace{\left[ \begin{array}{c|cc} a_j & s_j & \mathbf{0} \\ s_j & \mathbf{0} & A_{j+1} \end{array} \right]}_{\substack{1 \\ 1 \\ \vdots \\ n-j-1}}. \quad (5.44)$$

Тогда после  $k$  повторных применений леммы 5.3 имеем отвечающую этому моменту процесса итоговую матрицу преобразований

$$Q^{(k)} = \begin{bmatrix} I_k & 0 \\ 0 & Q_{k+1} \end{bmatrix} Q^{(k-1)}, \quad 1 \leq k \leq N = n - 2, \quad Q^{(0)} = I_n \quad (5.45)$$

и промежуточный результат тридиагонализации данной матрицы  $A$  в виде

$$Q^{(k)} A (Q^{(k)})^T = \begin{bmatrix} a_1 & s_1 & & & & & & \\ s_1 & a_2 & s_2 & & & & & \\ & s_2 & \ddots & \ddots & & & & \\ & & \ddots & a_k & s_k & & & \\ & & & & s_k & & & \\ & & & & & \mathbf{0} & & \\ & & & & & & \mathbf{0} & \\ & & & & & & & A_{k+1} \end{bmatrix}.$$

### Приведение квадратной матрицы к двухдиагональному виду

Применим к произвольной квадратной матрице слева преобразование  $Q_1$  и справа преобразование  $S_2$  (беря любое из них как преобразование Хаусхолдера или как преобразование Гивенса), при этом  $Q_1$  выберем из задачи желаемого преобразования ведущего столбца и  $S_2$  – из задачи желаемого преобразования ведущей строки, а именно: при действии  $Q_1$  – получение ненулевого диагонального элемента и нулевых элементов ниже него в первом (ведущем) столбце; при действии  $S_2$  – сохранение диагонального элемента, получение в смежной с ним

позиции ненулевого элемента и нулевых элементов правее него в первой (ведущей) строке.

**Лемма 5.4.** Пусть дана матрица  $A = A(n, n)$ . Тогда существуют ортогональное преобразование  $Q_1$  (Хаусхолдера или Гивенса) и ортогональное преобразование  $S_2$  (Хаусхолдера или Гивенса) такие, что

$$Q^{(1)}AS^{(1)} = \begin{matrix} & \overbrace{\hspace{1cm}}^1 & \overbrace{\hspace{1cm}}^1 & \overbrace{\hspace{1cm}}^{n-2} \\ & s_1 & a_1 & \mathbf{0} \\ \begin{matrix} 1 \\ n-2 \end{matrix} \left\{ \begin{bmatrix} \mathbf{0} & \tilde{A} \end{bmatrix} \right. \end{matrix}, \quad \begin{cases} Q^{(1)} = Q_1, \\ S^{(1)} = \begin{bmatrix} I_1 & \mathbf{0} \\ \mathbf{0} & S_2 \end{bmatrix}. \end{cases} \quad (5.46)$$

**Теорема 5.4** (Бидиагонализация квадратной матрицы). Пусть дана квадратная матрица  $A = A(n, n)$ ,  $A_1 := A$  и для каждого  $j = 1, \dots, k$ , где  $k \leq n-2$ , выбраны элементарное преобразование  $Q_j$  (Хаусхолдера типа  $T_j$  или Гивенса типа  $P_j$ ) и элементарное преобразование  $S_{j+1}$  (Хаусхолдера типа  $T_{j+1}$  или Гивенса типа  $P_{j+1}$ ) таким образом, что в результате получаем

$$Q_j A_j \left[ \begin{array}{c|c} I_1 & \mathbf{0} \\ \hline \mathbf{0} & S_{j+1} \end{array} \right] = \underbrace{\begin{matrix} 1 \\ \vdots \\ n-j \end{matrix}}_{\text{ }} \left[ \begin{array}{ccc} 1 & 1 & n-j-1 \\ \overbrace{s_j \quad a_j}^{} & \mathbf{0} & \overbrace{\quad \quad \quad}^{} \\ \mathbf{0} & & A_{j+1} \end{array} \right]. \quad (5.47)$$

Тогда после  $k$  повторных применений леммы 5.4 имеем отвечающие этому моменту процесса итоговые матрицы преобразований

$$\left. \begin{aligned} Q^{(k)} &= \begin{bmatrix} I_{k-1} & 0 \\ 0 & Q_k \end{bmatrix} Q^{(k-1)}, \quad k \leq n-2, \quad Q^{(0)} = I_n, \quad Q^{(1)} = Q_1, \\ S^{(k)} &= S^{(k-1)} \begin{bmatrix} I_k & 0 \\ 0 & S_{k+1} \end{bmatrix}, \quad k \leq n-2, \quad S^{(1)} = \begin{bmatrix} I_1 & 0 \\ 0 & S_2 \end{bmatrix} \end{aligned} \right\} \quad (5.48)$$

и промежуточный результат бидиагонализации данной матрицы  $A$  в виде

$$Q^{(k)} A S^{(k)} = \begin{bmatrix} s_1 & a_1 \\ s_2 & a_2 \\ \vdots & \vdots \\ s_k & a_k \\ \hline 0 & A_{k+1} \end{bmatrix}$$

Выполнив после  $k = n - 2$  ещё одно левостороннее преобразование  $Q_{n-1}$  (что отвечает применению верхней формулы (5.48) для  $k = n - 1$ ), получаем окончательно

$$Q^{(n-1)} A S^{(n-2)} = \begin{bmatrix} s_1 & a_1 & & & \\ & s_2 & a_2 & & \\ & & \ddots & \ddots & \\ & & & s_{n-1} a_{n-1} & \\ 0 & & & & s_n \end{bmatrix}.$$

Основное применение указанных двусторонних ортогональных преобразований заключается в вычислении сингулярных значений [13] произвольной матрицы  $A = A(n, n)$ , а также в решении проблемы собственных значений [30]. Однако эти преобразования можно использовать и для решения системы линейных алгебраических уравнений  $Ax = f$ . После приведения матрицы к двух- или трёхдиагональному виду система уравнений легко решается. Например, в случае с трёхдиагональной матрицей система очень эффективно решается методом прогонки [5].

## 5.11 Ортогонализация Грама–Шмидта

Пусть  $A = A(m, n)$  – матрица, имеющая  $m$  строк и  $n$  столбцов, причём  $m \geq n$ . Обозначая  $i$ -й столбец через  $a_i$ , запишем  $A = [a_1, a_2, \dots, a_n]$ ,  $a_i \in \mathbb{R}^m$ . Рассмотрим случай матрицы полного ранга, т. е.,  $\text{rank } A = n$ . Тогда набор векторов  $\{a_i\}$  порождает некоторое подпространство  $\mathcal{L} \in \mathbb{R}^m$ , т. е., может считаться его базисом. Назовём этот набор исходным базисом и преобразуем его в ортонормированный базис. Такое преобразование называется *процедурой ортогонализации системы векторов*  $\{a_1, a_2, \dots, a_n\}$ .

Согласно определению, ортонормированным базисом в  $\mathcal{L} \in \mathbb{R}^m$  называется система векторов  $\{q_1, q_2, \dots, q_n\}$  такая, что

- 1)  $\forall i : q_i \in \mathbb{R}^m$ ,  $m \geq n$ ,  $q_i^T q_i = \|q_i\|^2 = 1$ ;
- 2)  $\forall i, j, i \neq j : q_i^T q_j = 0$

и любой вектор  $a_i$  имеет единственное представление

$$a_i = \sum_{j=1}^n q_j b_{ji}, \quad i = 1, 2, \dots, n,$$

где  $b_i^T = (b_{1i}, b_{2i}, \dots, b_{ni})$  – вектор-строка некоторых коэффициентов. Следовательно, матрицу  $A$  можно представить в виде произведения двух матриц  $A = QB$ , где  $Q = [q_1, q_2, \dots, q_n]$  – матрица размера  $(m \times n)$ , составленная из столбцов  $q_i \in \mathbb{R}^m$ , а  $B = [b_1, b_2, \dots, b_n]$  – матрица размера  $(n \times n)$ , составленная из столбцов  $b_i \in \mathbb{R}^n$ . Матрица  $Q = Q(m, n)$  в этом представлении состоит из ортонормированных векторов-столбцов, в частном случае  $m = n$  в качестве  $Q$  имеем ортогональную матрицу, т. е.,  $Q^T Q = I$ .

Таким образом, ортогонализация столбцов матрицы  $A$  есть представление  $A = QB$ , где  $Q$  – матрица тех же размеров, что и  $A$ , но в отличие от  $A$ , имеющая ортонормированные столбцы, при этом  $B$  – квадратная матрица, обеспечивающая равенство  $A = QB$ . Очевидно, существует бесконечное множество таких представлений матрицы  $A$ , поскольку число ортонормированных базисов не ограничено. Для обеспечения единственности среди множества версий  $A = QB$  выберем представление, при котором  $B$  – треугольная матрица, которую далее традиционно будем обозначать  $R$ , поскольку в ней оказывается заполнен правый (*right*) верхний угол, т. е.,  $R = R_{\text{ne}}$ . Хотя традиционно ортогонализацией Грама–Шмидта называют отыскание по матрице  $A$  такой матрицы  $Q$ , что  $A = QR$ , где  $R = R_{\text{ne}}$ , для  $R$  будем допускать все четыре возможных варианта заполнения (см. подразд. 5.8):

- ◆ вариант 1:  $R = R_{\text{ne}}$ , где  $R_{\text{ne}}$  – верхняя правая треугольная матрица;
- ◆ вариант 2:  $R = R_{\text{sw}}$ , где  $R_{\text{sw}}$  – нижняя левая треугольная матрица;
- ◆ вариант 3:  $R = R_{\text{se}}$ , где  $R_{\text{se}}$  – нижняя правая треугольная матрица;
- ◆ вариант 4:  $R = R_{\text{nw}}$ , где  $R_{\text{nw}}$  – верхняя левая треугольная матрица.

Для ортогонализации системы векторов вычисление матрицы  $R$  в явном виде может и не требоваться, хотя такое вычисление всегда присутствует. Ниже, рассматривая ортогонализацию Грама–Шмидта обобщённо, т. е., во всевозможных вариантах треугольного заполнения матрицы  $R$ , мы будем требовать явного нахождения факторов (сомножителей)  $Q$  и  $R$  в разложении  $A = QR$ . Для любого из вариантов возможны три формы алгоритма, отличающиеся порядком действий.

### • Грама–Шмидта Ортогонализация (ГШО)

Этот вариант алгоритма предполагает вычисление ненулевых элементов матрицы  $R$  по столбцам, начиная с самого короткого (одноэлементного) столбца.

- **Модифицированная ГШО (МГШО)**

В этом варианте ненулевые элементы матрицы  $R$  вычисляются по строкам, начиная с самой длинной (состоящей из  $n$  элементов) строки.

- **МГШО с выбором ведущего вектора**

Этот вариант МГШО-алгоритма использует стратегию выбора ведущего вектора. В качестве очередного, подлежащего ортогонализации вектора, выбирается тот из оставшихся столбцов матрицы  $A$ , который имеет наибольшую длину (евклидову норму). Хотя эта стратегия требует дополнительных вычислительных затрат, в некоторых плохо обусловленных задачах она так же полезна, как и выбор главного элемента в методе Гаусса.

Таким образом, данной темой – ортогонализация Грама–Шмидта – в предлагаемом проекте (см. подразд. 5.16) охвачено  $(4 \times 3) = 12$  различных вариантов задачи разложения  $A = QR$ .

**Замечание 5.7.** Обратим внимание на различия между двумя типами ортогональных преобразований, а именно: между преобразованиями Хаусхолдера и Гивенса (если говорить о левосторонних их версиях, хотя возможны и правосторонние) – это один тип преобразований, и ортогонализацией Грама–Шмидта – другой тип. Первый тип обеспечивает равенства (5.39), (5.40), (5.41) или (5.42), где преобразованная матрица  $QA$  имеет тот же размер, что и исходная матрица  $A$ , и в её составе присутствует блок, появляющийся как треугольная матрица  $R$  в одном из четырёх углов этой матрицы  $QA$ . При этом матрица  $Q$  ортогонального преобразования – квадратная. В случае ортогонализации Грама–Шмидта имеем  $A = QR$ , где  $Q$  – матрица того же размера, что и  $A$ , но с ортонормированными столбцами, а  $R$  – строго квадратная матрица с треугольным заполнением.

## 5.12 Алгоритмы ортогонализации Грама–Шмидта

Рассмотрим задачу  $QR$ -разложения матрицы  $A(m, n)$ ,  $m \geq n$ , полного ранга на основе ортогонализации Грама–Шмидта.

В данной задаче, рассматривая пример  $n = 3$ , имеем

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ & r_{22} & r_{23} \\ & & r_{33} \end{bmatrix},$$

$A = [a_1, a_2, a_3] = [q_1, q_1, q_3]R = [r_{11}q_1, r_{12}q_1 + r_{22}q_2, r_{13}q_1 + r_{23}q_2, r_{33}q_3]$ . В результате получаем линейную систему

$$\begin{aligned} r_{11}q_1 &= a_1, \\ r_{12}q_1 + r_{22}q_2 &= a_2, \\ r_{13}q_1 + r_{23}q_2 + r_{33}q_3 &= a_3. \end{aligned}$$

Ясно, что каждое из этих выражений представляет собой разложение вектора  $a_1$ ,  $a_2$  или  $a_3$  по системе ортов  $\{q_1, q_2, q_3\}$ , при этом коэффициент  $r_{ij}$  есть алгебраическая проекция вектора  $a_j$  на орт  $q_i$ .

В силу треугольности матрицы  $R$  эта система легко решается. Из первого уравнения находим орт  $q_1$  вдоль вектора  $a_1$  и координату (проекцию как число) первого вектора  $a_1$  вдоль орта  $q_1$ :

$$q_1 = a_1 / \|a_1\|, \quad r_{11} = \|a_1\|.$$

Второе уравнение есть разложение вектора  $a_2$  на сумму проекций вдоль ортов  $q_1$  и  $q_2$ . Так как орт  $q_1$  уже найден, то координата  $r_{12}$  легко определяется в виде

$$r_{12} = a_2^T q_1.$$

После этого из второго уравнения имеем

$$r_{22}q_2 = a_2 - r_{12}q_1,$$

следовательно,

$$\begin{aligned} q_2 &= (a_2 - r_{12}q_1) / \|a_2 - r_{12}q_1\|, \\ r_{22} &= \|a_2 - r_{12}q_1\|. \end{aligned}$$

**Замечание 5.8.** По предположению,  $\text{rank } A = n$ , т. е., ни один из векторов  $a_i$  не является нулевым, и все  $a_i$  образуют линейно независимую систему. Поэтому  $r_{11} \neq 0$ ,  $r_{22} \neq 0$  и  $r_{33} \neq 0$ , следовательно, существует  $R^{-1}$ .

Продолжая решение системы, для третьего уравнения находим

$$r_{13} = a_3^T q_1, \quad r_{23} = a_3^T q_2$$

и затем определяем

$$r_{33}q_3 = a_3 - r_{13}q_1 - r_{23}q_2.$$

Отсюда

$$\begin{aligned} q_3 &= (a_3 - r_{13}q_1 - r_{23}q_2) / \|a_3 - r_{13}q_1 - r_{23}q_2\|, \\ r_{33} &= \|a_3 - r_{13}q_1 - r_{23}q_2\|. \end{aligned}$$

Таким образом, получили классический метод ГШО, отличающийся тем, что матрица  $R$  определяется по столбцам с номерами  $k = 1, 2, \dots, n$ .

В настоящее время существуют две более эффективные версии — ортогонализации Грама–Шмидта:

- Алгоритм МГШО (модифицированная схема).
- Алгоритм МГШО с выбором ведущего вектора.

Первые две версии — классическая и модифицированная — показаны здесь — на стр. 194, а модифицированная с выбором ведущего вектора — на стр. 195.

### Алгоритм ГШО (классическая схема)

Для  $k = 1$  до  $n$

$$r_{ik} = a_k^T q_i, \quad i = 1, 2, \dots, k - 1,$$

$$v = a_k - \sum_{i=1}^{k-1} r_{ik} q_i,$$

$$r_{kk} = (v^T v)^{1/2},$$

$$q_k = v / r_{kk}.$$

### Алгоритм МГШО (модифицированная схема)

Для  $k = 1$  до  $n$

$$r_{kk} = \|a_k\| = (a_k^T a_k)^{1/2},$$

$$a_k = a_k / r_{kk},$$

Для  $j = k + 1$  до  $n$

$$r_{kj} = a_j^T a_k,$$

$$a_j = a_j - r_{kj} a_k.$$

## Алгоритм МГШО с выбором ведущего вектора

```

Для  $k = 1$  до  $n$ 
     $q(k) = k,$ 
Для  $k = 1$  до  $n$ 
    Для  $s = k$  до  $n$ 
        Найти №  $l$ , для которого  $\|a_{q(l)}\| = \max_s \|a_{q(s)}\|$ ,
        Переставить номера:  $q(k) \leftrightarrow q(l)$ ,
         $r_{q(k),q(k)} = \|a_{q(k)}\| = (a_{q(k)}^T a_{q(k)})^{1/2}$ ,
         $a_{q(k)} = a_{q(k)} / r_{q(k),q(k)}$ ,
    Для  $j = k + 1$  до  $n$ 
         $r_{q(k),q(j)} = a_{q(j)}^T a_{q(k)}$ ,
         $a_{q(j)} = a_{q(j)} - r_{q(k),q(j)} a_{q(k)}$ .

```

Первая из более современных версий, называемая МГШО (Rice, 1966 [12]), отличается порядком вычислений матрицы  $R$ . В этой версии матрица  $R$  определяется по строкам с номерами  $k = 1, 2, \dots, n$ . Этот алгоритм требует меньше оперативной памяти, так как в нём не используется промежуточный вектор  $v$ . Кроме того, матрица  $A$  заменяется матрицей  $Q$ , потому что после операции деления имеем  $a_k = q_k$ . Одним из его преимуществ является то, что в него легко внедрить процедуру выбора ведущего столбца.

Чтобы получить вторую из более современных версий — так называемую МГШО с выбором ведущего вектора, — нужно изменить алгоритм МГШО таким образом, чтобы очередным ортогонализируемым вектором оказался не  $k$ -й, а тот, чья норма наибольшая среди всех оставшихся  $s$ -х векторов от  $s = k$  до  $s = n$ . Как и в подразд. 3.2, реально переставляются не столбцы матрицы  $A$ , а обмениваются значениями только элементы дополнительного вектора  $q$ , в котором фиксируются номера столбцов матрицы  $A$ . Доступ к элементам матрицы  $A$  осуществляется с использованием этого вектора. Перед началом работы основного алгоритма ортогонализации этот вектор перестановок  $q$  заполняется числами от 1 до  $n$  в естественном порядке нумерации столбцов матрицы  $A$ .

## 5.13 Решение систем после ортогонализации

1. Пусть дана система линейных алгебраических уравнений с квадратной невырожденной матрицей  $Ax = f$ . Тогда после ортогонального приведения матрицы с помощью одной из версий ортогонализации Грама–Шмидта имеем представление этой матрицы в виде  $A = QR$  и, следовательно,  $QRx = f$  и  $Rx = Q^T f$ .

2. Пусть дана система линейных алгебраических уравнений с прямоугольной матрицей  $A(m, n)$ ,  $m > n$ , полного ранга. Такая система называется *переопределённой системой*. *Нормальное псевдорешение*  $\bar{x}$ , найденное по методу наименьших квадратов (МНК), удовлетворяет *нормальным уравнениям*

$$A^T A \bar{x} = A^T f.$$

Поскольку  $A = QR$  и  $Q^T Q = I$ , эти уравнения эквивалентны уравнению

$$R \bar{x} = Q^T f,$$

которое совпадает по виду с уравнением из п. 1.

Чтобы вычислить  $x$  (для п. 1) или  $\bar{x}$  (для п. 2), находят вектор  $f' = Q^T f$ , а затем решают систему с треугольной матрицей  $R$  (методом подстановки).

## 5.14 Обращение матриц после ортогонализации

Для матрицы  $A = A(n, n)$  имеем  $A = QR$ , где  $Q = Q(n, n)$ . Отсюда

$$A^{-1} = R^{-1} Q^{-1} = R^{-1} Q^T.$$

Следовательно,  $A^{-1}$  есть решение матричного уравнения  $RX = Q^T$ . Чтобы найти  $i$ -й столбец матрицы  $A^{-1}$ , надо в качестве правой части взять  $i$ -й столбец матрицы  $Q^T$  и решить систему с треугольной матрицей  $R$  (как в подразд. 5.13 или подробнее в подразд. 5.6).

## 5.15 Задание на лабораторный проект № 3

Написать и отладить программу, реализующую ваш вариант ортогонального преобразования для численного решения систем линейных алгебраических уравнений  $Ax = f$  с квадратной матрицей  $A$ , вычисления  $\pm(\det(A))$  и  $A^{-1}$ . Предусмотреть предупреждение о невозможности решения указанных задач из-за присутствия (почти) линейно зависимых векторов среди столбцов матрицы

$A$  (в пределах ошибок округления ЭВМ или другого, заранее определённого критерия). Отделить основные части программы:

- а) подпрограмму факторизации матрицы  $A$ , отвечающую вашему варианту метода ортогонального приведения;
- б) подпрограмму решения систем линейных алгебраических уравнений;
- в) подпрограмму вычисления определителя матриц;
- г) подпрограмму обращения матриц;
- д) сервисные подпрограммы.

Уделить особое внимание эффективности программы (в смысле экономии оперативной памяти и скорости решения указанных выше задач). Предусмотреть пошаговое выполнение алгоритма ортогонального приведения с выводом результата на экран. Выполнить следующие пункты задания:

1. Провести подсчёт фактического количества операций, выполняемых при решении системы линейных алгебраических уравнений (отдельно число операций сложения, число операций умножения, число операций деления и число операций извлечения квадратного корня) и сравнить эти числа с теоретическими (оценочными) числами.

2. Оценить скорость решения задач, т. е., определить время, затраченное на решение системы линейных алгебраических уравнений, и время, затраченное на обращение матриц. Для этого спроектировать и провести эксперимент, который охватывает матрицы порядка от 10 до 100 (через 10 порядков). Представить результаты в виде таблицы и графика зависимости времени выполнения (в минутах и секундах) от порядка матриц. Таблицу и график вывести на экран.

3. Оценить точность решения систем линейных алгебраических уравнений, имеющих тот же самый порядок, что и задачи из п. 2. Для этого сгенерировать случайные матрицы  $A$ , выбрать точное решение  $x^*$  и образовать правые части  $f = Ax^*$ . Провести анализ точности вычисленного решения  $x$  от порядка матрицы. Результаты представить в виде таблицы и графика.

Для заполнения матрицы  $A$  использовать случайные числа из диапазона от  $-100$  до  $100$ . В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)$ , где  $n$  — порядок матрицы. Для оценки точности использовать норму вектора

$$\|x\|_\infty = \max_i(|x_i|).$$

4. Повторить п. 3 задания для плохо обусловленных матриц (см. подразд. 3.6 лабораторной работы № 1), имеющих порядок от 4 до 40.

5. Системы из пп. 2 и 3 необходимо решить двумя методами: методом исключения из лабораторной работы № 1 и методом ортогонального приведения

из лабораторной работы № 3. Сравнить точность решения и затраты машинного времени. Результаты представить в виде таблицы и графика.

6. Вычислить матрицу  $A^{-1}$  двумя способами:

1) через решение системы  $AX = I$  на основе метода исключения Гаусса из лабораторной работы № 1 (в соответствии со своим вариантом);

2) через решение системы  $AX = I$  на основе метода ортогонального преобразования (в соответствии со своим вариантом).

Сравнить затраты машинного времени и точность обращения способами 1) и 2). Эксперименты провести для матриц порядков от 10 до 100 через 10. Для оценки точности в обоих способах воспользоваться формулой из лабораторной работы (проекта) № 1.

## 5.16 Варианты задания на лабораторный проект № 3

По теме «Ортогональные преобразования матриц» студентам предлагается выполнение лабораторной работы – индивидуального проекта № 3.

Задание на этот проект содержит 28 вариантов, которые приведены в табл. 5.2 (см. стр. 199).

Все варианты различаются по следующим признакам:

- четыре варианта заполнения треугольной матрицы  $R$ ;
- три вида ортогональных преобразований:
  - 1) отражения Хаусхолдера,
  - 2) вращения Гивенса,
  - 3) ортогонализация Грама–Шмидта,
- две разновидности алгоритма ортогонализации по методу Хаусхолдера и по методу Гивенса:
  - 1) столбцово-ориентированный алгоритм,
  - 2) строчно-ориентированный алгоритм,
- три разновидности ортогонализации по методу Грама–Шмидта:
  - 1) классическая схема,
  - 2) модифицированная схема,
  - 3) модифицированная схема с выбором ведущего вектора.

Если нет других указаний преподавателя, выбирайте ваш вариант в табл. 5.2 (см. с. 199) по вашему номеру в журнале студенческой группы.

Таблица 5.2. Варианты задания на лабораторный проект № 3

Вариант заполнения матрицы $R$	Отражения Хаусхолдера		Вращения Гивенса		Ортогонализация Грама–Шмидта		
	a	b	a	b	c	d	e
 $\triangleq R_{ne}$	1	2	3	4	5	6	7
 $\triangleq R_{nw}$	8	9	10	11	12	13	14
 $\triangleq R_{se}$	15	16	17	18	19	20	21
 $\triangleq R_{sw}$	22	23	24	25	26	27	28

<sup>a</sup> – столбцово-ориентированный алгоритм;

<sup>b</sup> – строчно-ориентированный алгоритм;

<sup>c</sup> – классическая схема;

<sup>d</sup> – модифицированная схема;

<sup>e</sup> – модифицированная схема с выбором ведущего вектора.

## 5.17 Методические рекомендации для проекта № 3

Рассмотрим все пункты задания с точки зрения их реализации в проекте.

- Система меню для взаимодействия пользователя с программой.
- Функция факторизации матрицы.
- Функция решения системы линейных алгебраических уравнений.
- Функция вычисления определителя матрицы.
- Функция обращения матрицы через решение системы  $AX = I$  на основе метода ортогонального преобразования (в соответствии со своим вариантом).
- Эксперимент 1 «Подсчёт количества арифметических операций».
- Эксперимент 2 «Решение СЛАУ со случайными матрицами».
- Эксперимент 3 «Решение СЛАУ для плохо обусловленных матриц».
- Эксперимент 4 «Обращение матриц».

## Система меню для взаимодействия пользователя с программой

Действуйте так же, как вы строили меню на с. 115 при реализации **п. 1** из перечня заданий для проекта № 1 на с. 114.

Предусмотреть предупреждение о невозможности решения указанных задач из-за присутствия (почти) линейно зависимых векторов среди столбцов матрицы  $A$  (в пределах ошибок округления ЭВМ или другого, заранее определённого критерия).

### Функция факторизации матрицы

Рассмотрим возможную реализацию функции факторизации на примере столбцово-ориентированного алгоритма отражения Хаусхолдера для варианта заполнения матрицы  $R_{ne}$  (описание алгоритма см. в подразд. 5.5, с. 175).

#### Листинг 5.1.

```
float [,] A = new float[m,n] ;
Random r = new Random();
for (int i=0;i<m;i++)
for (int j=0;j<n;j++)
    A[i,j]=r.Next(-10,10); //генерируем матрицу A
float [] s = new float [Math.Min(m-1,n)]; //генерируем вектор s
for (int k=0;k<Math.Min(m-1,n);k++){
    // вычисляем s_k
    for (int t=k;t<m;t++)
        s[k]=-Math.Sign(A[k,k])*Math.Sqrt(A[t,k]*A[t,k]);
    // вычисляем u (u замещает столбцы из нижней треугольной
    // части матрицы A)
    A[k,k]=A[k,k]-s[k];
    // вычисляем \alpha
    float d=1/(s[k]*A[k,k]);
    for (int j=k+1;k<n;k++) {
        %считаем сумму
        float sum_uA = 0;
        for (int i=k;i<m;i++)
            sum_uA=sum_uA+A[i,k]*A[i,j];
        // считаем \lambda
        float l=d*sum_uA;
```

```

        for (int i=k;i<m;i++)
            A[i,j]=A[i,j]+l*A[i,k];
    }
}

```



Особенность программной реализации заключается в том, что в результате работы алгоритма в строго верхней треугольной части матрицы  $A$  получаем элементы матрицы  $R$ , в строке  $s$  получаем диагональные элементы матрицы  $R$ , а в нижней треугольной части матрицы  $A$  сохраняются векторы  $u(k)$ .

## Функция решения системы линейных алгебраических уравнений

Если известно разложение матрицы  $A = QR$ , тогда решение СЛАУ  $Ax = b$  сводится к решению эквивалентной системы  $Rx = Q^Tb$ . Сначала найдём произведение  $Q^Tb = b_1$ . Для этого применим к вектору  $b$  цепочку преобразований, которая была проделана над столбцами матрицы  $A$ . (см. листинг 5.1).

### Листинг 5.2.

```

int [] b1 = new int[b.Length];
for (int i=0;i<b.Length;i++)
b1[i]=b[i];
// A - преобразованная после факторизации матрица
for (int k=0;k<Math.Min(m-1,n);k++) {
    // вычисляем \alpha
    float d=1/(s[k]*A[k,k]);
    // считаем сумму
    float sum_uA = 0;
    for (int i=k;i<m;i++)
        sum_uA=sum_uA+A[i,k]*b1[i];
    // считаем \lambda
    float l=d*sum_uA;
    for (int i=k;i<m;i++)
        b1[i]=b1[i]+l*A[i,k];
}

```



Далее найдём искомый вектор  $x$ , решая треугольную систему  $Rx = b_1$ .

### Листинг 5.3.

```
for (int i=n-1;i>-1;i--) {  
    int sum=0;  
    if (i<n)  
        for (int j=i+1;j<n;j++)  
            sum=sum+R[i,j]*x[j];  
    x[i]=(b1[i]-sum)/R[i,i];  
}
```



### Функция вычисления определителя матрицы

Если нам известно разложение матрицы  $A = QR$ , тогда определитель квадратной матрицы  $A$  посчитаем по формуле:

$$\det A = \pm \det R = \pm \prod_{i=1}^n R_{ii}.$$

### Функция обращения матрицы через решение системы $AX = I$

Обращение матрицы  $A$  через решение системы  $AX = I$  следует выполнять на базе того метода ортогонального преобразования, который вам задан (т. е., в соответствии со своим вариантом).

Для матрицы  $A = A(n, n)$  имеем  $A = QR$ . Отсюда  $A^{-1} = R^{-1}Q^{-1} = R^{-1}Q^T$ . Следовательно,  $A^{-1}$  есть решение матричного уравнения  $RX = Q^T$ . Чтобы найти  $i$ -й столбец матрицы  $A^{-1}$ , надо в качестве правой части взять  $i$ -й столбец единичной матрицы, применить к нему заданную цепочку преобразований и решить систему с треугольной матрицей  $R$ .

### Эксперимент 1 «Подсчёт количества арифметических операций»

Написать реализацию первого эксперимента для исследования количества операций. Описание эксперимента можно найти в подразд. 5.15 на с. 196.

#### Методика проведения эксперимента:

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

**Число вычислительных операций:**

Порядок матрицы	Теоретическое число операций			Реальное число операций		
	а	б	в	а	б	в

где а, б, в обозначают количество операций: а – сложение, б – умножение и деление (вместе), в – извлечение квадратного корня. Каждая строка в таблице будет зависеть от  $n$  – количества уравнений в системе. Число  $n$  должно изменяться от 10 до 100 через 10 порядков, т. е., в таблице должно быть 10 строк.

2. Присвоить  $n$  очередное значение.
3. Сгенерировать матрицу  $A$  случайным образом (см. лабораторный проект № 1).
4. Выполнить факторизацию матрицы  $A$  в соответствии с п. 2 списка заданий на с. 199.
5. Найти решение СЛАУ  $x$  в соответствии с п. 3 списка заданий на с. 199.
6. Подсчитать теоретическое (оценочное) число операций сложения, число операций умножения, число операций деления и число операций извлечения квадратного корня.
7. Подсчитать реальное число операций сложения, число операций умножения, число операций деления и число операций извлечения квадратного корня с помощью специальных счётчиков, которые вы добавите в функции факторизации и решения.
8. Добавить в таблицу полученные экспериментальные данные: порядок  $n$ , число операций сложения, число операций умножения, число операций деления и число операций извлечения квадратного корня.

**Эксперимент 2 «Решение СЛАУ со случайными матрицами»**

Написать реализацию второго эксперимента для исследования решения СЛАУ со случайными матрицами. Описание эксперимента можно найти в подразд. 5.15 на с. 196.

В данном эксперименте сравниваются два способа решения СЛАУ со случайными матрицами:

- первый способ – по методу решения систем в проекте № 1,
- второй способ – по методу решения систем в проекте № 3.

**Методика проведения эксперимента:**

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

**Решение СЛАУ со случайными матрицами:**

Порядок матрицы	Время		Погрешность	
	способ 1	способ 2	способ 1	способ 2

Каждая строка в таблице будет зависеть от  $n$  – количества уравнений в системе. Число  $n$  должно изменяться от 10 до 100 через 10 порядков, т. е., в таблице будет 10 строк.

2. Присвоить  $n$  очередное значение.
3. Заполнить матрицу  $A$  случайными числами и сгенерировать вектор  $b$ .
4. Выполнить факторизацию матрицы  $A$  в соответствии с п. 2 списка заданий на с. 199.
5. Найти решение СЛАУ  $x$  в соответствии с п. 3 списка заданий на с. 199.
6. Выполнить факторизацию матрицы  $A$  в соответствии с п. 2 списка заданий на с. 114 (из лабораторной работы № 1).
7. Найти решение СЛАУ  $x$  в соответствии с п. 3 списка заданий на с. 114 (из лабораторной работы № 1).
8. Подсчитать время решения задачи как сумму времени факторизации матрицы и времени решения СЛАУ для двух методов: метода исключения в лабораторном проекте № 1 и метода ортогонального разложения в лабораторном проекте № 3.
9. Оценить погрешность решения СЛАУ для двух методов: метода исключения в лабораторном проекте № 1 и метода ортогонального разложения в лабораторном проекте № 3:
  - Задать точное решение  $x^*$ . В качестве точного решения нужно взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – порядок матрицы.
  - Образовать правые части  $b := Ax^*$ , матрица  $A$  известна из п. 3.

- Найти решение СЛАУ  $Ax = b$ .
- Вычислить погрешность решения СЛАУ как максимальный модуль разности компонента вектора точного решения и вектора найденного решения.

Добавить в таблицу и файл для хранения экспериментальных данных полученные экспериментальные данные: порядок  $n$ , время выполнения, погрешность решения СЛАУ, теоретическое и реальное число операций.

10. Пункты 2–9 повторить для всех значений  $n$ .
11. Построить следующие графики решения СЛАУ (для двух методов):
  - зависимость времени решения от порядка матриц;
  - зависимость погрешности решения от порядка матриц.

При построении графиков необходимо использовать данные из файла с экспериментальными данными.

12. Проанализировать полученные данные, сравнить погрешность решения и затраты машинного времени для методов в лабораторном проекте № 1 и лаборатории проекте № 3. Сделать содержательные выводы об эффективности и качестве реализованных численных методов.

### **Эксперимент 3 «СЛАУ с плохо обусловленными матрицами»**

Написать реализацию третьего эксперимента для исследования погрешности решения СЛАУ в случае, когда матрица  $A$  является плохо обусловленной. Описание эксперимента можно найти в подразд. 5.15 на с. 196. Перед проведением эксперимента написать функцию, вычисляющую погрешность решения.

В данном эксперименте сравниваются два способа решения СЛАУ с плохо обусловленными матрицами:

- первый способ – по методу решения систем в проекте № 1,
- второй способ – по методу решения систем в проекте № 3.

#### **Методика проведения эксперимента:**

Для каждой из 10 плохо обусловленных матриц на стр. 50, 51 выполнить следующее:

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

**Решение СЛАУ с плохо обусловленными матрицами:**

Порядок матрицы	Погрешность способ 1	Погрешность способ 2
--------------------	-------------------------	-------------------------

Каждая строка в таблице будет зависеть от  $n$  – количества уравнений в системе. Если матрица  $A$  имеет фиксированный размер (матрицы с номерами 2, 3, 6, 10), тогда  $n =$  размеру матрицы, и в таблице будет только одна строка. Если порядок матрицы не фиксирован (матрицы с номерами 1, 4, 5, 7, 8, 9), тогда число  $n$  должно изменяться от 4 до 40 через 4 порядка, т. е., в таблице будет 10 строк.

2. Присвоить  $n$  очередное значение.
3. Заполнить матрицу  $A$  в соответствии с её номером и сгенерировать вектор  $b$  в соответствии с п. 1 списка заданий на с. 199 (для заданной матрицы!).
4. Найти погрешность решения системы первым способом.
5. Найти погрешность решения системы вторым способом.
6. Добавить в таблицу и файл для хранения экспериментальных данных полученные экспериментальные данные: порядок  $n$ , погрешность решения СЛАУ.
7. Пункты 2–6 повторить для всех значений  $n$ .
8. Построить графики зависимости погрешности решения СЛАУ от порядка матрицы.
9. Проанализировать полученные данные и сделать содержательные выводы об эффективности и качестве реализованных численных методов.

#### **Эксперимент 4 «Обращение матриц»**

Написать реализацию четвёртого эксперимента для исследования скорости и погрешности процедур обращения матриц. Подробное описание эксперимента можно найти в подразд. 5.15, с. 196.

В данном эксперименте сравниваются два способа обращения матриц:

- первый способ – по методу решения систем в проекте № 1,
- второй способ – по методу решения систем, реализованному в проекте № 3.

Перед проведением эксперимента написать процедуру, вычисляющую погрешность найденной обратной матрицы. Формулы (3.21), (3.22) в разд. 3.7 позволяют оценить сверху погрешность обращения матрицы  $A$ . Для этого необходимо, в соответствии с этими формулами, сначала умножить исходную матрицу на найденную обратную и вычесть результат из единичной матрицы, затем найти норму найденной разности и поделить её на норму исходной матрицы. Норма матрицы вычисляется по формуле (3.22) на с. 111 – это максимальная сумма модулей элементов строк.

### **Методика проведения эксперимента:**

1. Вывести на экран «шапку» таблицы с экспериментальными данными:

#### **Обращение матриц:**

Порядок	Время		Погрешность	
	способ 1	способ 2	способ 1	способ 2

Каждая строка в таблице будет зависеть от  $n$  – размера матрицы. Число  $n$  должно изменяться от 10 до 100 через 10 порядков, т. е., в таблице будет 10 строк.

2. Присвоить  $n$  очередное значение.
3. Заполнить матрицу  $A$  случайными числами.
4. Найти обратную матрицу первым способом через решение системы  $AX = I$  в лабораторном проекте № 1 (в соответствии со своим вариантом).
5. Подсчитать время обращения матрицы первым способом.
6. Найти погрешность обращения матрицы первым способом.
7. Найти обратную матрицу вторым способом в соответствии с п. 5 перечня заданий на с. 199.
8. Подсчитать время обращения матрицы вторым способом.
9. Найти погрешность обращения матрицы вторым способом.

10. Добавить в таблицу полученные экспериментальные данные: порядок  $n$ , время и погрешность обращения первым и вторым способами.
11. Пункты 2–10 повторить для всех значений  $n$ .
12. Проанализировать полученные данные и сделать содержательные выводы об эффективности и качестве реализованных численных методов.

## 5.18 Тестовые задачи для проекта № 3

Используйте приводимые ниже задачи в двух режимах:

- для контроля собственного понимания алгоритма,
- для контроля правильности вашего программирования.

### Задача 1

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & 6 \\ -2 & 6 & -7 \\ -2 & 7 & 1 \end{pmatrix}$$

выполнить следующее:

- a. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).
- б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (5, -15, -8)^T$ .

- в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 2

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & 3 \\ -2 & 6 & -1 \\ -2 & 7 & 7 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).
- б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (6, 3, 12)^T$ .

- в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 3

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & -5 \\ -2 & 6 & 5 \\ -2 & 7 & -3 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).
- б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (8, -1, 8)^T$ .

- в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 4

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & 6 \\ -2 & 6 & 3 \\ -2 & 7 & -3 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (5, -5, -12)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 5

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & 5 \\ -2 & 6 & -5 \\ -2 & 7 & 3 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (6, 3, 12)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 6

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & 1 \\ -2 & 6 & 1 \\ -2 & 7 & 9 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-1, -3, 4)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 7

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & 7 \\ -2 & 6 & -9 \\ -2 & 7 & -1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-8, 1, -8)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 8

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & -3 \\ -2 & 6 & 1 \\ -2 & 7 & -7 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (6, 3, 12)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 9

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & -6 \\ -2 & 6 & 7 \\ -2 & 7 & -1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (7, 1, 10)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 10

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & -2 \\ -2 & 6 & -1 \\ -2 & 7 & -9 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (1, 3, -4)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 11

Для матрицы

$$A = \begin{pmatrix} 1 & 2 & -7 \\ -2 & 6 & 9 \\ -2 & 7 & 1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (10, -5, 4)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 12

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & -6 \\ -2 & 4 & -3 \\ -2 & 5 & 3 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (8, 9, 4)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 13

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & 3 \\ -2 & 4 & -1 \\ -2 & 5 & 7 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (5, 5, 14)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 14

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & -5 \\ -2 & 4 & 5 \\ -2 & 5 & -3 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-7, -1, -10)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 15

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & 6 \\ -2 & 4 & 3 \\ -2 & 5 & -3 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (2, 1, -6)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 16

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & 5 \\ -2 & 4 & -5 \\ -2 & 5 & 3 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-1, 7, 0)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 17

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & 2 \\ -2 & 4 & 1 \\ -2 & 5 & 9 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-4, -7, -16)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 18

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & 6 \\ -2 & 4 & -7 \\ -2 & 5 & 1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (8, -1, 8)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 19

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & 7 \\ -2 & 4 & -9 \\ -2 & 5 & -1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-8, 11, 3)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, x \in \mathbb{R}^3$ .

### Задача 20

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & -3 \\ -2 & 4 & 1 \\ -2 & 5 & -7 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-1, -3, 4)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 21

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & -6 \\ -2 & 4 & 7 \\ -2 & 5 & -1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (9, -3, 6)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 22

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & -2 \\ -2 & 4 & -1 \\ -2 & 5 & -9 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (-2, -6, -7)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 23

Для матрицы

$$A = \begin{pmatrix} 1 & 3 & -7 \\ -2 & 4 & 9 \\ -2 & 5 & 1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $QR$ -разложение матрицы  $A$  с помощью ортогональных преобразований (Хаусхолдера / Гивенса / ГШО / МГШО).

б. С помощью  $QR$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (2, 6, 7)^T$ .

в. С помощью  $QR$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

## 5.19 Заключение по разделу 5

В данном разделе рассмотрены стандартные алгоритмы  $QR$ -разложения матрицы, основанные на трёх типах ортогональных преобразований: отражения (Хаусхолдера), плоские вращения (Гивенса) и процедура ортогонализации Грама-Шмидта (последняя рассматривается в различных вариантах). Даны теоретические сведения, достаточные для выполнения проекта № 3.

В проекте № 3 предлагается выполнить программирование  $QR$ -разложения и на этой основе решить классические задачи вычислительной линейной алгебры: (1) найти решение СЛАУ, (2) найти обратную матрицу и (3) вычислить

её определитель. При этом матрицу системы предлагается формировать тремя различными способами: вводить «вручную» с клавиатуры компьютера, генерировать случайным образом или же из списка специальных – плохо обусловленных – матриц.

В числе задач, решаемых в проекте № 3, – сравнение времени и погрешности решения указанных выше задач (1) и (2).

Этот проект является базовым для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

# 6

## Пример программной реализации проекта № 1

### 6.1 Постановка задачи

#### Цели и задачи лабораторного проекта

При выполнении данного проекта мы преследуем три конкретные цели:

1. Развить навыки аналитического мышления. Эти навыки должны развиваться в процессе приобретения новых знаний из области вычислительной математики и новых умений решения конкретных задач.
2. Научиться решать базовые задачи вычислительной линейной алгебры. Эти умения будут нарабатываться в процессе решения тестовых задач «вручную». Такие решения потребуются для тестирования (отладки) самостоятельно разработанных компьютерных программ.
3. Приобрести реальный опыт разработки компьютерных программ высокого (почти профессионального) уровня и опыт профессионального применения компьютеров посредством написания, отладки и многочисленных прогонов своих программ. Приобретённый опыт будет проверяться посредством выполнения индивидуального задания на лабораторный проект.

#### Задание на лабораторный проект:

1. Реализовать  $\bar{LU}$ -разложение в гауссовом исключении по столбцам с выбором главного (ведущего) элемента по строке активной подматрицы (вариант № 2 из списка заданий).
2. Реализовать алгоритм решения системы линейных алгебраических уравнений (СЛАУ), используя результат  $\bar{LU}$ -разложения.

3. Реализовать вычисление обратной матрицы на основе  $\bar{LU}$ -разложения, многократно обращаясь к процедуре решения СЛАУ.
4. Реализовать алгоритм вычисления обратной матрицы, используя элементарные преобразования и формулу обращения  $A^{-1} = \bar{U}^{-1}\bar{L}^{-1}$ .
5. Разработать удобный пользовательский интерфейс.
6. Отчёты должны быть представлены в графической и табличной формах.
7. Все алгоритмы должны быть реализованы на языке C#. В качестве среды разработки использовать Visual Studio 13, .NET Framework 4.5.

**Замечание 6.1.** В оставшейся части разд. 6 мы будем демонстрировать ход разработки этого проекта. В листингах кода (старый код будет дописываться) новые фрагменты кода будут выделены специальным образом (это мы делаем для того, чтобы каждый раз не записывать весь код заново). ■

Например, нам нужно выделить вот такой новый фрагмент кода:

```
Console.WriteLine(txt);
```

Тогда мы искусственно «окружаем» этот фрагмент знаками: {# перед фрагментом и #} после него. Введённые знаки, естественно, надо удалить перед прогоном программы, – они здесь служат исключительно для придания листингам большей наглядности. Этот приём понятен из следующего примера 6.1.

### Пример 6.1.

```
void WriteText (string txt) {  
}  
// Ниже между {# и #} вставляем новый фрагмент кода:
```

```
void WriteText (string txt) {  
{#  
Console.WriteLine(txt);  
#}  
}
```



**Замечание 6.2.** В конце данного раздела имеется ссылка на готовый проект, а в конце каждого подраздела имеется ссылка на исходный код класса, описываемого в этом подразделе. ■

## 6.2 Класс с реализацией алгоритмов

Для решения поставленной задачи в первую очередь необходимо написать класс, в котором будут реализованы алгоритмы  $LU$ -разложения, решения СЛАУ и отыскания обратной матрицы, а также должен вестись подсчёт количества операций, затрачиваемых на каждый из алгоритмов. Хотя программа и должна иметь удобный пользовательский интерфейс, для простоты тестирования и отладки работы класса будем использовать консольный проект. Позднее реализованный класс можно будет скомпилировать в отдельную библиотеку и подключить к проекту или просто скопировать исходный код класса в нужное место.

После создания нового проекта создаём новый класс. В листинге ниже приведён код пустого проекта.

### Листинг 6.1.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NumMeth
{
    class Program {
        static void Main(string[] args) {

        }
    }
}
```



В следующем листинге приведён код пустого класса, в котором будут реализованы все необходимые алгоритмы.

### Листинг 6.2.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace NumMeth {
    public class NumMeth {
        public NumMeth() {
        }
    }
}
```



В методе Main класса Program нужно создать экземпляр пока ещё пустого класса NumMeth:

### Листинг 6.3.

```
static void Main(string[] args)
{
{#
NumMeth meth = new NumMeth();
#}
Console.ReadKey();
}
```



Вернёмся к классу NumMeth. В нём необходимо завести поле, хранящее матрицу, и метод, через который эту матрицу можно будет передать из основного кода программы. Кроме того, в данном методе можно будет реализовать  $LU$ -разложение. Сначала напишем  $LU$ -разложение без выбора ведущего элемента.

В листинге ниже описаны алгоритмы  $LU$ -разложения и вывод итоговой матрицы в виде строки, что необходимо для вывода матрицы в консоль. Поля double[,] A и int N, должны быть определены в классе заранее. В метод setA в качестве аргумента передаётся матрица, которую необходимо факторизовать, и размер самой матрицы. Метод getA возвращает матрицу в строковом формате – в нашем случае для вывода в консоль.



**Листинг 6.4.**

```

public void setA(double[,] a, int n){
    N = n;
    A = new double[n,n];
    saveA = a;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i, j] = a[i, j];
        }
    }
    for (int k = 0; k < n; k++){
        for (int j = k + 1; j < n; j++){
            A[k,j] /= A[k,k];
        }
        for (int i = k + 1; i < n; i++){
            for (int j = k + 1; j < n; j++){
                A[i,j] -= A[i,k] * A[k,j];
            }
        }
    }
}
public string getA() {
    string result = "";
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result += Math.Round(A[i, j], 3).ToString() + "\t";
        }
        result += "\r\n";
    }
    return result;
}

```

Настало время протестировать написанную программу. Для этого в методе Main необходимо создать матрицу. Можно использовать свою.

Для тестирования выбрана следующая матрица:

$$\begin{bmatrix} 2 & 4 & -4 & 6 \\ 1 & 4 & 2 & 1 \\ 3 & 8 & 1 & 1 \\ 2 & 5 & 0 & 5 \end{bmatrix}.$$

Для неё запишем следующий код:

### Листинг 6.5.

```
static void Main(string[] args)
{
    NumMeth meth = new NumMeth();
    //{
    double[,] a = new double[,] {{2,4,-4,6}, {1,4, 2,1},
    {3,8, 1,1}, {2,5, 0,5}};
    meth.setA(a);
    Console.WriteLine(meth.getA());
    //}
    Console.ReadKey();
}
```



Нужно протестировать написанную процедуру факторизации матрицы.

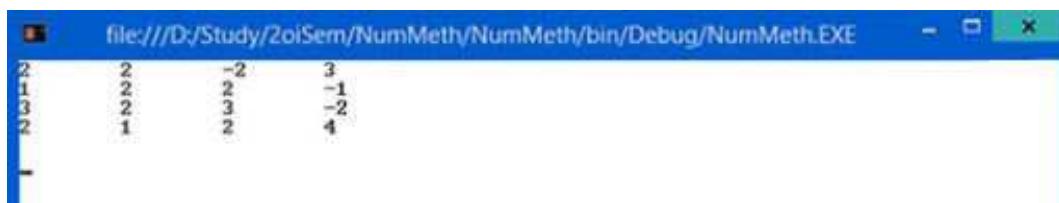


Рис. 6.1. Вывод матрицы после компиляции процедуры факторизации

Вывод правильный. Можно двигаться дальше.

На следующем шаге необходимо реализовать выбор ведущего элемента. Для того чтобы не переставлять столбцы местами каждый раз, когда будет найден ведущий элемент, заведём массив перестановок столбцов длиной, равной размеру матрицы. Изначально значения каждого элемента будут равны его индексу в массиве. Каждый раз при нахождении ведущего элемента, перестановка будет

совершаться не в матрице, а в массиве перестановок. Необходимо также завести глобальную для класса переменную `znak`, по умолчанию равную единице. При каждой перестановке знак этой переменной будет меняться на противоположный (последствием умножения на  $-1$ ). Значение этой переменной потребуется при вычислении определителя матрицы.

В начале тела класса `NumMeth` создадим массив целочисленного типа и целочисленную переменную `znak`:

```
int[] p_c;
int znak = 1;
```

В начале метода `setA` необходимо определить размер массива `p_c` и заполнить его начальными значениями:

### Листинг 6.6.

```
{#
p_c = new int[N];
for (int i = 0; i < N; i++) {
p_c[i] = i;
}
#}
```



После этого к элементам матрицы  $A$  будем обращаться не напрямую по номерам строки и столбца элемента, а через элементы массива `p_c`. Например, чтобы обратиться к элементу, стоящему в строке  $i$  и столбце  $j$ , будем обращаться не к  $A[i, j]$ , а к  $A[i, p_c[j]]$ .

После такой модернизации необходимо немного переписать весь метод `setA`, изменив способ обращения к элементам матрицы  $A$ :

### Листинг 6.7.

```
for (int k = 0; k < n; k++)
{
    for (int j = k + 1; j < n; j++) {
        A[{#k,p_c[j]#}] /= A[{#k,p_c[k]#}];
    }
    for (int i = k + 1; i < n; i++) {
        for (int j = k + 1; j < n; j++) {
```

```

    A[{#i,p_c[j]#}] -= A[{#i,p_c[k]#}] * A[{#k,p_c[j]#}];  

}
}  

}

```



Теперь обеспечим выбор ведущего элемента. Ведущим будем считать наибольший по абсолютному значению элемент, находящийся в текущей строке. Таким образом мы уменьшим вероятность деления на ноль в строке  $A[k, p_c[j]] / A[k, p_c[k]]$  листинга 6.7, а поскольку такая вероятность всё равно имеется, обрабатаем ситуацию, когда максимальный элемент по абсолютному значению не превышает заданный эпсилон (машинный ноль).

Но сначала о перестановках. Перед циклом, в котором элементы строки делятся на ведущий элемент, необходимо выполнить его поиск:

### Листинг 6.8.

```

{#
int imax = k;
double max = Math.Abs(A[k, p_c[k]]);
for (int i = k + 1; i < n; i++) {
if (Math.Abs(A[k, p_c[i]]) > max) {
max = Math.Abs(A[k, p_c[i]]);
imax = i;
}
}
if (imax != k) {
int buf = p_c[k];
p_c[k] = p_c[imax];
p_c[imax] = buf;
znak*=-1;
}
#}

// В это место далее будет вставлен код проверки на равенство
// главного элемента нулю

for (int j = k + 1; j < n; j++) {

```

```
A[k,p_c[j]] /= A[k,p_c[k]];
}
```

Для обработки ситуации, при которой произойдёт деление на ноль, необходимо завести переменную *флаг*, которой будет присвоено значение `true`, если максимальный по абсолютному значению элемент не превышает эпсилон (этот флаг также следует определить где-нибудь в начале класса):

```
public bool flagError = false;
```

Вычислять машинный эпсилон будем так, как показано ниже в листинге 6.9.

### Листинг 6.9.

```
double eps;
double a = 1;
do { eps = a; a /= 2; } while (a != 0);
```

После выполнения этого кода в переменную `eps` будет записано минимальное по абсолютному значению возможное число, представляемое типом `double`. Для того чтобы каждый раз при создании класса `NumMeth` не вычислять значение эпсилон, его можно вычислить в основном теле программы, а в класс передавать в качестве аргумента конструктора. Для этого в классе `NumMeth` создадим новое поле `EPS` и новый конструктор:

```
double EPS = 0;
public NumMeth(double e) EPS = e;
```

Код для вычисления эпсилон необходимо вставить в метод `Main` класса `Program`, а также изменить запись для создания экземпляра класса `NumMeth`:

### Листинг 6.10.

```
static void Main(string[] args) {
{#
double eps;
double a = 1;
do { eps = a; a /= 2; } while (a != 0);
```

```
NumMeth meth = new NumMeth(eps);
#
double[,] a = new double[,] {{2,4,-4,6},
                             {1,4, 2,1},
                             {3,8, 1,1},
                             {2,5, 0,5}};
meth.setA(a);
{#
if (num.flagError) {
    Console.WriteLine("LU-разложение невозможno,
                      произошло деление на ноль!");
    return;
}
#
Console.WriteLine(meth.getA());
Console.ReadKey();
}
```



Так как метод `setA` у одного и того же экземпляра класса будет вызываться несколько раз с разными матрицами, то сбрасывать значение переменной `flagError` необходимо каждый раз при вызове этого метода. Ниже в листинге реализовано сравнение ведущего элемента с эпсилон (эта проверка вставлена вместо комментария «В это место далее будет вставлен код проверки на равенство главного элемента нулю»):

### Листинг 6.11.

```
public void setA(double[,] a, int n){
{#
flagError = false;
if (Math.Abs(A[k, p_c[k]]) < 2*EPS) {
// Сравнение ведущего элемента с эпсилон
    flagError = true;
    return;
}
#}
```



Если ведущий элемент равен нулю (а он максимальный по абсолютному значению), значит и все остальные элементы также равны нулю. В таком случае можно вернуть сообщение об ошибке и выйти из метода. Для проверки можно попытаться передать в качестве аргумента нулевую матрицу размера 3:

### Листинг 6.12.

```
{#
double[,] zero = new double[,] {{0,0,0},
{0,0,0},
{0,0,0}};
meth.setA(zero)
#}
```

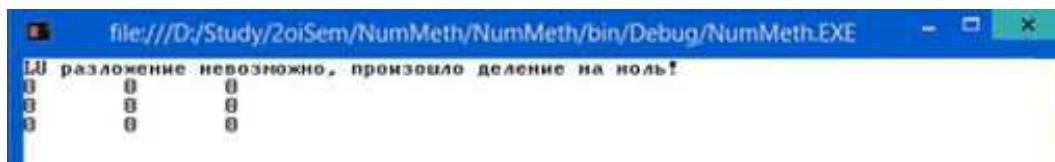


Рис. 6.2. Вывод ошибки

Реализовать все алгоритмы – это лишь половина задачи. Помимо реализации, необходимо фиксировать время их выполнения и количество операций, затраченных на их решение. Начать решать эту задачу необходимо сейчас. Для этого объявим в теле класса две переменные: одна – целочисленная для подсчёта количества операций, другая – вещественная для фиксации затраченного времени на выполнение алгоритма.

### Листинг 6.13.

```
int oper_f = 0;
double time_f = 0;
public double TIME { get { return time_f; } }
public int OPER_F { get { return oper_f; } }
```

Переменные `oper_f` и `time_f` должны быть доступны из всех методов внутри класса `NumMeth`. Из других классов значения этих переменных должны

быть доступны только для чтения. Помимо фактического времени и числа операций, потребуется считать теоретическое время. Для этого создадим в классе NumMeth дополнительную целочисленную переменную и «гетер» для неё:

### Листинг 6.14.

```
int oper_t = 0;  
public int OPER_T { get { return oper_t; } }
```



Теперь в каждом такте цикла будем увеличивать oper\_f на единицу, а значение переменной oper\_t будем считать в начале методов, в зависимости от длины массива или матрицы.

Пока не двинулись дальше метода setA, сразу реализуем подсчёт времени выполнения и количества операций: теоретического и фактического. Для того чтобы посчитать время выполнения, в начале метода setA необходимо зафиксировать время начала выполнения, а в конце метода setA зафиксировать время окончания выполнения. Разность между ними нас интересует. В начале метода setA зафиксируем дату и посчитаем теоретическое число операций:

### Листинг 6.15.

```
DateTime date = DateTime.Now;  
oper_t = n * n * n;
```



В конце метода setA вычтем из текущей даты сохранённую дату и сохраним количество миллисекунд в созданную переменную time:

### Листинг 6.16.

```
TimeSpan sp = DateTime.Now - date;  
times =sp.TotalMilliseconds;
```



В каждом внутреннем цикле необходимо вставить строку  
oper\_f++;

Далее приведём исходный код изменённого метода setA класса NumMeth.

**Листинг 6.17.**

```

// Метод выполняет LU-разложение
public void setA(double[,] a, int n) {
    // Аргументы: a - матрица, n - размер матрицы.
    flagError = false; // Сбрасываем флаг ошибки, которая может
                      // возникнуть в результате LU-разложения
    oper_f = 0;        // Фактическое число оперций
    oper_t = 0;        // Теоретическое число операций
    DateTime date = DateTime.Now; // Сохраняем время перед разложением
    N = n;            // Сохраняем размер матрицы в глобальную
                      // переменную (так он нужен в других методах)
    A = a;
    oper_t = n * n * n; // Теоретическая сложность LU-разложения
                      // имеет ценку  $O(N^3)$ .
    p_c = new int[n]; // Создаём массив перестановок
    for (int i = 0; i < n; i++) {
        // Заполняем массив перестановок начальными значениями
        p_c[i] = i;
    }
    for (int k = 0; k < n; k++) { // Начало LU-разложения
        int imax = k; // Сохраняем индекс текущего элемента
        double max = Math.Abs(A[k, p_c[k]]); // Сохраняем значение
        // текущего элемента (с учётом перестановок)
        for (int i = k + 1; i < n; i++) {
            // В этом цикле выполняется поиск максимального по абсолютному
            // значению элемента активной подматрицы
            if (Math.Abs(A[i, p_c[k]]) > max) {
                max = Math.Abs(A[i, p_c[k]]); // В случае нахождения
                // элемента, большего по абсолютному значению, чем текущий элемент,
                // запоминаем его значение и номер столбца
                imax = i;
            }
        }
        if (imax != k) { // Выполняем перестановку и изменяем знак
            // в переменной, на которую будет умножаться определитель матрицы
            int buf = p_c[k];
            p_c[k] = p_c[imax];
        }
    }
}

```

```
p_c[imax] = buf;
znak *= -1;
oper_f++;
}
if (Math.Abs(A[k, p_c[k]]) < 2*EPS) {
// Сравниваем значение максимального элемента в активной подматрице
// с двумя машинными эпсилонами, если значение меньше,
// то устанавливаем флаг ошибки и завершаем LU-разложение
flagError = true;
return;
}
for (int j = k + 1; j < n; j++) {
// В этом цикле выполняется нормировка строки
A[k, p_c[j]] /= A[k, p_c[k]];
oper_f++;
}
for (int i = k + 1; i < n; i++) {
// Вычитание из всех строк активной подматрицы текущей строки
for (int j = k + 1; j < n; j++) {
A[i, p_c[j]] -= A[i, p_c[k]] * A[k, p_c[j]];
oper_f++;
}
}
}
}
// Фиксируем время окончания LU-разложения и сохраняем его
TimeSpan sp = DateTime.Now - date;
time_f = sp.TotalMilliseconds;
}
```



Теперь метод `setA` реализован полностью. Пора приступать к основным задачам, которые класс `NumMeth` должен выполнять, а именно:

- решение СЛАУ,
- обращение матриц двумя способами
- и вычисление определителя матрицы, а также
- вычисление погрешности всех операций и подсчёт числа операций.

### Начнём с решения СЛАУ

Общий вид СЛАУ:  $Ax = b$ , где  $A$  – матрица коэффициентов,  $x$  – вектор неизвестных,  $b$  – вектор с известными элементами. Матрицу  $A$  мы представляем (в нашем варианте проекта) в виде произведения матриц  $L$  и  $U$ . Благодаря этому, система примет следующий вид:  $L\bar{U}x = b$ . Произведение  $\bar{U}x$  представим как дополнительный (неизвестный) вектор  $y \triangleq \bar{U}x$ . Тогда, зная  $b$ , найдём решение  $x$  СЛАУ в два этапа: сначала вычислим вектор  $y$  (процедура прямой подстановки в системе  $Ly = b$ )

$$y_i = \left( b_i - \sum_{j=1}^{i-1} a_{ij} * y_j \right) / a_{ii}, \quad i = 1, 2, \dots, n,$$

а потом в процессе обратной подстановки в системе  $\bar{U}x = y$  получаем искомое решение  $x$ :

$$x_i = y_i - \sum_{j=i+1}^n a_{ij}x_j, \quad i = n-1, \dots, 1, \quad x_n = y_n.$$

Это те же формулы, что и (3.5), (3.6) на с. 89, но записанные здесь с учётом того важного факта, что все вычисления во время разложения матрицы  $A = L\bar{U}$  производятся в том же самом массиве, где до их начала находилась заданная матрица системы  $A$ .

Остаётся только реализовать эти формулы на C#. Для этого создадим дополнительный метод в классе NumMeth. Назовём его `getX`. В качестве аргумента в этот метод будем передавать вектор  $b$ , а возвращать метод будет искомый вектор  $x$ . Вывод вектора  $x$  будем производить с учётом перестановок, для чего надо реализовать ещё один дополнительный метод, который эти перестановки выполняет.

### Листинг 6.18.

```
// Метод решения СЛАУ
public double[] getX(double[] B) {
    // Аргументы: B - вектор (известная, правая часть СЛАУ)
    double[] X = new double[B.Length];
    // Создаём вектор X (неизвестная часть СЛАУ).
    DateTime date = DateTime.Now;
    // Сохраняем время начала решения СЛАУ
    oper_t += N * N;
```

```
// Увеличиваем теоретическое число операций на N^2
// (такова теоретическая сложность этого алгоритма)
    for (int i = 0; i < B.Length; i++) {
        // Вычисляем вектор Y
        X[i] = B[i];
        for (int k = 0; k <= i - 1; k++) {
            X[i] -= A[i, p_c[k]] * X[k];
            oper_f++;
        }
        X[i] /= A[i, p_c[i]];
        oper_f++;
    }
    for (int i = N - 1; i >= 0; i--) {
        // Вычисляем вектор X.
        for (int k = i + 1; k < N; k++) {
            X[i] -= A[i, p_c[k]] * X[k];
            oper_f++;
        }
    }
}

// Вычисляем разницу во времени до и после решения СЛАУ
TimeSpan sp = DateTime.Now - date;
// Увеличиваем время на эту разницу
time_f += sp.TotalMilliseconds;
// Возвращаем вектор X с учётом перестановок.
return Preobraz(X);
}

// Вспомогательный метод, возвращающий вектор X с учётом перестановок
private double[] Preobraz(double[] x) {
// Аргументы: X - вектор (неизвестная часть СЛАУ)
    double[] X = new double[N];
    for (int i = 0; i < N; i++) {
        X[p_c[i]] = x[i];
    }
    return X;
}
```



В приведённом выше коде векторы  $x$ ,  $y$  и  $b$  обозначены заглавными символами, соответственно,  $X$ ,  $Y$  и  $B$ .

В первом цикле вычисляется вектор  $Y$ , все значения записываются в вектор  $X$ . Во втором цикле вычисляются значения вектора  $X$ , также в него записываются и новые значения. Такой подход позволяет обойтись всего одним массивом для вычисления значений, хотя по формулам их должно быть два. (Вообще говоря, рекомендуется обходиться и одним массивом: сначала в него вводят  $B$ , затем в него сохраняют  $Y$  и, наконец, в него же записывают  $X$ ). Кроме того можно заметить, что в методе `getX` изменяются значения фактического и теоретического числа операций, а также времени выполнения.

С процедурой решения СЛАУ закончили. Настало время проверить работу метода. В качестве массива  $B$  передадим вектор  $(1,2,3,4)$ . Посчитав вручную, находим, что вектор  $X = (2.94, -0.94, 1.13, 0.56)$ . Этот результат нам потребуется для отладки программы решения.

Для того, чтобы программа вывела в консоль элементы вектора  $X$ , необходимо в методе `Main` класса `Program` дописать следующий код:

### Листинг 6.19.

```
meth.setA(a);
{
    double[] X = meth.getX(new double[] { 1,2,3,4 });
    string result = "";
    for (int i = 0; i < X.Length; i++) {
        result += X[i].ToString() + "\r\n";
    }
    Console.WriteLine(result);
}
```

■  
Вывод должен быть как на рис. 6.3.

Теперь соберём всё, что уже реализовано, и сделаем красивый вывод. Выведем матрицу  $A$  после  $LU$ -разложения, значения вектора  $X$ , теоретическое число операций, фактическое число операций и общее затраченное время выполнения. Все эти данные у нас есть, нужно только оформить вывод (листинг 6.20):

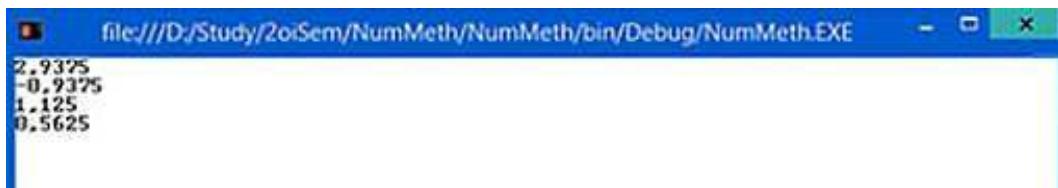


Рис. 6.3. Вывод решения СЛАУ

**Листинг 6.20.**

```

meth.setA(a);
double[] X = meth.getX(new double[] { 1,2,3,4 });
Console.WriteLine(meth.getA());
string result = "";
for (int i = 0; i < X.Length; i++) {
    result += X[i].ToString() + "\r\n";
}
Console.WriteLine(result);
{#
Console.WriteLine("Теоретическое число операций:
    " + meth.OPER_T.ToString());
Console.WriteLine("Фактическое число операций:
    " + meth.OPER_F.ToString());
Console.WriteLine("Время: " + meth.TIME.ToString());
#}

```

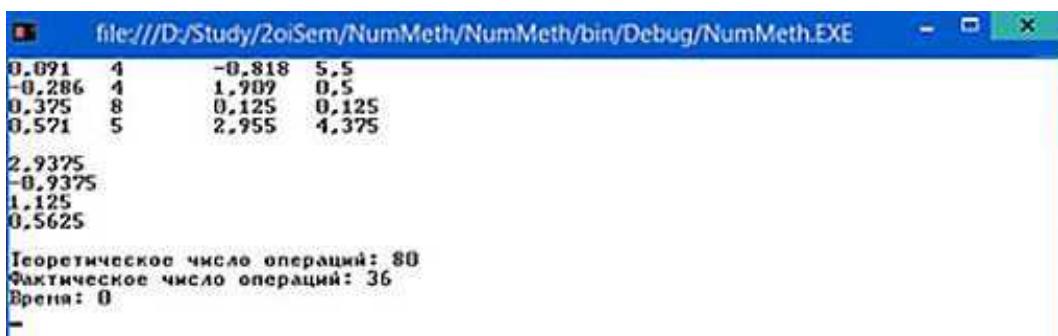


Рис. 6.4. Вывод числа и времени операций

Перед тем, как переходить к вычислению обратной матрицы, можно написать метод, который будет возвращать определитель матрицы.

Определителем матрицы  $A$  будет произведение элементов, стоящих на главной диагонали матрицы  $L$  после  $LU$ -разложения. Здесь также необходимо учитывать число перестановок, за это отвечает переменная `znak`:

$$|A| = \text{znak} * \prod_{i=1}^n l_{ii}.$$

Остается только написать метод в классе `NumMeth` и выполнить проверку:

### Листинг 6.21.

```
public double getDet() {
    double r = 1;
    for (int i = 0; i < n; i++) {
        r *= A[i, p_c[i]];
    }
    return znak*r;
}
```



В методе `Main` класса `NumMeth` после вывода времени, затраченного на выполнение, в новой строке сделаем вывод определителя (рис. 6.5):

### Листинг 6.22.

```
Console.WriteLine("Время: " + meth.TIME.ToString());
{#
Console.WriteLine("Определитель матрицы:" +
    meth.getDet().ToString());
#}
Console.ReadKey();
```



Теперь перейдём к нахождению обратной матрицы через решение СЛАУ.

Имеем  $AA^{-1} = I$ . Это равносильно системе нескольких СЛАУ (их количество равно  $n$ ):

$$\begin{cases} Aa_1^{-1} = e_1 \\ \dots \\ Aa_n^{-1} = e_n \end{cases}$$

где  $e_i$  –  $i$ -й столбец единичной матрицы  $I$ .

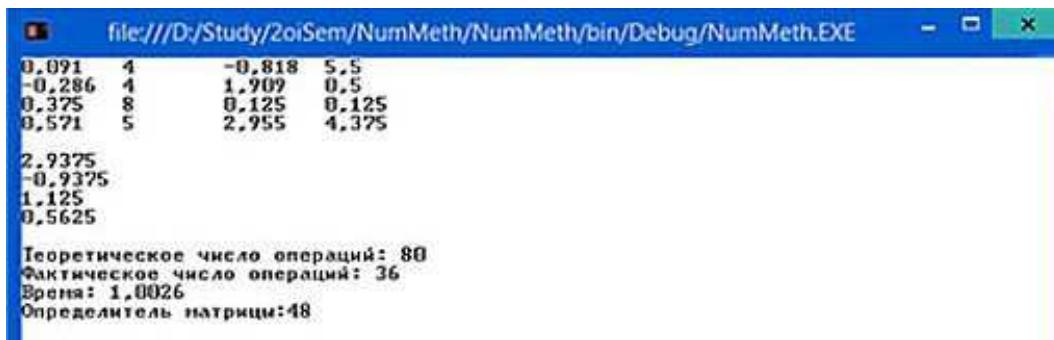


Рис. 6.5. Вывод определителя матрицы

Остаётся только найти векторы-столбцы  $a_1^{-1}$ ,  $a_2^{-1}$ , ...,  $a_n^{-1}$  и сформировать из них матрицу, которая и будет являться обратной. Для решения задачи воспользуемся уже созданным программным кодом, как это показано ниже в листинге 6.23.

### Листинг 6.23.

```
// Метод вычисления обратной матрицы, через решение СЛАУ
public double[,] Inv1(double[,] A, int n) {
    // Аргументы: A - матрица, n - размер матрицы
    setA(A, n);      // Выполняем LU-разложение
    if (flagError) return null;
    // Если в результате LU-разложения произошло деление на ноль,
    // то выходим из метода
    double[][] X = new double[n] [];
    // Создаём вложенные массивы (X - обратная матрица,
    // E - единичная матрица)
    double[][] E = new double[n] [];

    // Для решения СЛАУ нужно передавать вектор, а это возможно
    // только с вложенными массивами
    for (int i = 0; i < N; i++) {
        // Заполняем единичную матрицу значениями
        E[i] = new double[N];
        E[i][i] = 1;
    }
    for (int i = 0; i < N; i++) {
        // Решение СЛАУ
```

```

X[i] = new double[n];
X[i] = getX(E[i]);
oper_f++;
}
DateTime date = DateTime.Now;

// В решение СЛАУ уже ведётся учёт времени, следовательно,
// в этом методе его необходимо начинать только сейчас
for (int i = 0; i < n; i++) {
// Выполняем транспонирование матрицы, так как решение
// записывалось в строки, записывать значения сразу
// в столбцы нельзя,
    for (int j = i; j < n; j++) {
// так как работали с вложенными массивами
// Работать с обычными массивами также не совсем удобно,
// так как пришлось бы записывать значения каждый раз
// после решения СЛАУ,
        double s = X[i][j];      // в предыдущем цикле
// Таким образом, оценка сложности осталась бы той же,
// а кода было бы больше
        X[i][j] = X[j][i];
        X[j][i] = s;
        oper_f++;
    }
}
TimeSpan sp = DateTime.Now - date;
time_f += sp.TotalMilliseconds;
return toMatrix(X);
// Переводим вложенный массив в обычный двухмерный
// и возвращаем
}

// Метод преобразует вложенный квадратный массив
// в квадратный двумерный массив
private double[,] toMatrix(double[][] x) {
// Аргументы: x - вложенный массив
double[,] X = new double[N, N];

```

```

for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        X[i, j] = x[i][j];
    }
}
return X;
}

```



Метод Inv1 принимает в качестве аргумента исходную матрицу. Внутри метода сначала выполняется  $\bar{LU}$ -разложение, затем происходит генерация единичной матрицы, и далее каждая строка обратной матрицы заполняется элементами, которые являются решением СЛАУ. В методе Inv1 обратная матрица представлена вложенным двумерным массивом. Такое представление позволило сразу присваивать результат решения строкам матрицы. Также следует заметить, что в итоге обратная матрица получилась транспонированная, поэтому перед выводом необходимо поменять строки и столбцы местами.

### *Обращение матрицы через элементарные преобразования*

Если  $A = \bar{LU}$ , то справедливо и следующее:  $A^{-1} = \bar{U}^{-1}\bar{L}^{-1}$ . Остаётся только найти обратные матрицы для  $\bar{L}$  и  $\bar{U}$ . Это сделать гораздо проще и быстрее, чем для  $A$ , так как в треугольных матрицах  $\bar{L}$  и  $\bar{U}$  известны (уже вычислены) заведомо нулевые элементы. После обращения будет необходимо найти произведение треугольных матриц  $\bar{U}^{-1}$  и  $\bar{L}^{-1}$ . Тем самым мы найдём матрицу, обратную матрице  $A$  (перед выводом необходимо вернуть столбцы на свои места).

### **Листинг 6.24.**

```

// Метод вычисления обратной матрицы через элементарные
// преобразования
// (A^(-1) = L^(-1)*U^(-1)).
public double[,] Inv2(double[,] a, int n) {
    setA(a, n);
    double sum = 0;
    DateTime date = DateTime.Now;
    oper_t += N * N * N;

```

```

// Первый этап - подготовка (обращение элементарных матриц)
for (int i = 0; i < N; i++) {
    for (int j = i + 1; j < N; j++) {
        A[i, p_c[j]] *= -1;
    }
}
for (int j = 0; j < N; j++) {
    A[j, p_c[j]] = 1 / A[j, p_c[j]];
    oper_f++;
    for (int i = j + 1; i < N; i++) {
        oper_f++;
        A[i, p_c[j]] = -A[i, p_c[j]] * A[j, p_c[j]];
    }
}
// Считаем матрицу U^(-1)
for (int k = n - 1; k > 0; k--) {
    for (int i = 0; i < k - 1; i++) {
        for (int j = k; j < N; j++) {
            oper_f++;
            A[i, p_c[j]] += A[i, p_c[k - 1]] * A[k - 1, p_c[j]];
        }
    }
}
// Считаем матрицу L^(-1)
for (int k = 0; k < N - 1; k++) {
    for (int i = k + 2; i < N; i++) {
        for (int j = 0; j <= k; j++) {
            oper_f++;
            A[i, p_c[j]] += A[i, p_c[k + 1]] * A[k + 1, p_c[j]];
        }
    }
    for (int j = 0; j <= k; j++) {
        oper_f++;
        A[k + 1, p_c[j]] = A[k + 1, p_c[j]] * A[k + 1, p_c[k + 1]];
    }
}
// Перемножение матриц U^(-1) и L^(-1)

```

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        if (i < j) {
            sum = 0;
            for (int k = j; k < N; k++) {
                sum += A[i, p_c[k]] * A[k, p_c[j]];
                oper_f++;
            }
        }
        else if (i >= j) {
            sum = A[i, p_c[j]];
            for (int k = i + 1; k < N; k++) {
                sum += A[i, p_c[k]] * A[k, p_c[j]];
                oper_f++;
            }
        }
        A[i, p_c[j]] = sum;
        oper_f++;
    }
}
// Выполняем обратную перестановку
double[,] R = new double[N, N];
int[] p2 = new int[N];
for (int i = 0; i < N; i++) {
    p2[p_c[i]] = i;
    oper_f++;
}
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        R[i, j] = A[p2[i], p_c[j]];
        oper_f++;
    }
}
TimeSpan sp = DateTime.Now - date;
time_f += sp.TotalMilliseconds;
return R;
}
```



### *Вычисление погрешности расчётов*

Чтобы вычислить погрешность решения СЛАУ, необходимо изначально знать точное решение. Оно, конечно, неизвестно. Однако мы пойдём от обратного: создадим вектор  $X_{\text{точ}}$  с известными значениями. С помощью него вычислим вектор  $B$  путём умножения матрицы  $A$  на вектор  $X_{\text{точ}}$ . В классе NumMeth необходимо создать следующий метод для вычисления вектора  $B$ :

#### **Листинг 6.25.**

```
// Метод возвращает правую часть СЛАУ, вычисленную
// по известному вектору X
public double[] getB(double[,] matrix, double[] X, int n) {
    // Аргументы: matrix - матрица, X - точное решение СЛАУ,
    // n - размер матрицы.
    double[] B = new double[n];
    for (int i = 0; i < n; i++) B[i] = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            B[i] += matrix[i, j] * X[j];
        }
    }
    return B;
}
```



На основе полученных данных найдём решение СЛАУ, вектор  $X$ . После этого сравним два вектора. Наибольшая (по абсолютной величине) разность между соответствующими элементами и будет погрешность.

#### **Листинг 6.26.**

```
// Метод вычисления погрешности решения СЛАУ

public double VectorPogr(double[] X1, double[] X2) {

    // Аргументы: два вектора, один из которых представляет собой точное
    // решение СЛАУ, другой - приближённое решение.

    double result = 0;
    for (int i = 0; i < X1.Length; i++) {
```

```
    if (Math.Abs(X1[i] - X2[i]) > result) {
        result = Math.Abs(X2[i] - X1[i]);
    }
}
return result;
}
```



Чтобы оценить погрешность вычисления обратной матрицы, необходимо найти произведение матрицы на обратную, результатом произведения должна стать матрица, приближённая к единичной. После этого необходимо сравнить этот результат с единичной матрицей, наибольшая (по абсолютной величине) разность между соответствующими элементами и будет погрешность. Метод, вычисляющий погрешность, также реализован в классе NumMeth.

### Листинг 6.27.

```
// Метод возвращает погрешность вычисления обратной матрицы.
public double PogreshInv(double[,] A, double[,] invA, int n) {
// Аргументы: A - исходная матрица, invA - обратная матрица,
// n - размер матрицы
double result = 0;

// Переменная будет хранить значение максимальной погрешности.
// Создаём единичную матрицу и заполняем её значениями.
double[,] E = new double[n, n];
for (int i = 0; i < n; i++) {
    E[i, i] = 1;
}

// Умножение матрицы на обратную даёт единичную матрицу.
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        double rr = 0;
        for (int k = 0; k < n; k++) {
            rr += A[i, k] * invA[k, j];
        }
        if (Math.Abs(E[i, j] - rr) > result)
            result = Math.Abs(E[i, j] - rr);
    }
}
```

```

// Если разность между соответствующими элементами больше по
// абсолютному значению текущей максимальной погрешности,
// то обновляем значение погрешности.
}
}

return result;
}

```

Построение класса NumMeth завершено. Осталось решить, кто и как будет им пользоваться в дальнейшем. Можно просто скопировать весь исходный код в свой проект и компилировать вместе со всем проектом, а можно сейчас скомпилировать из этого класса dll библиотеку и в итоговом проекте подключать эту библиотеку. Воспользуемся вторым способом. Для создания библиотеки необходимо создать из класса NumMeth dll новый проект «библиотека классов» и скопировать туда наш класс NumMeth (рис. 6.6).

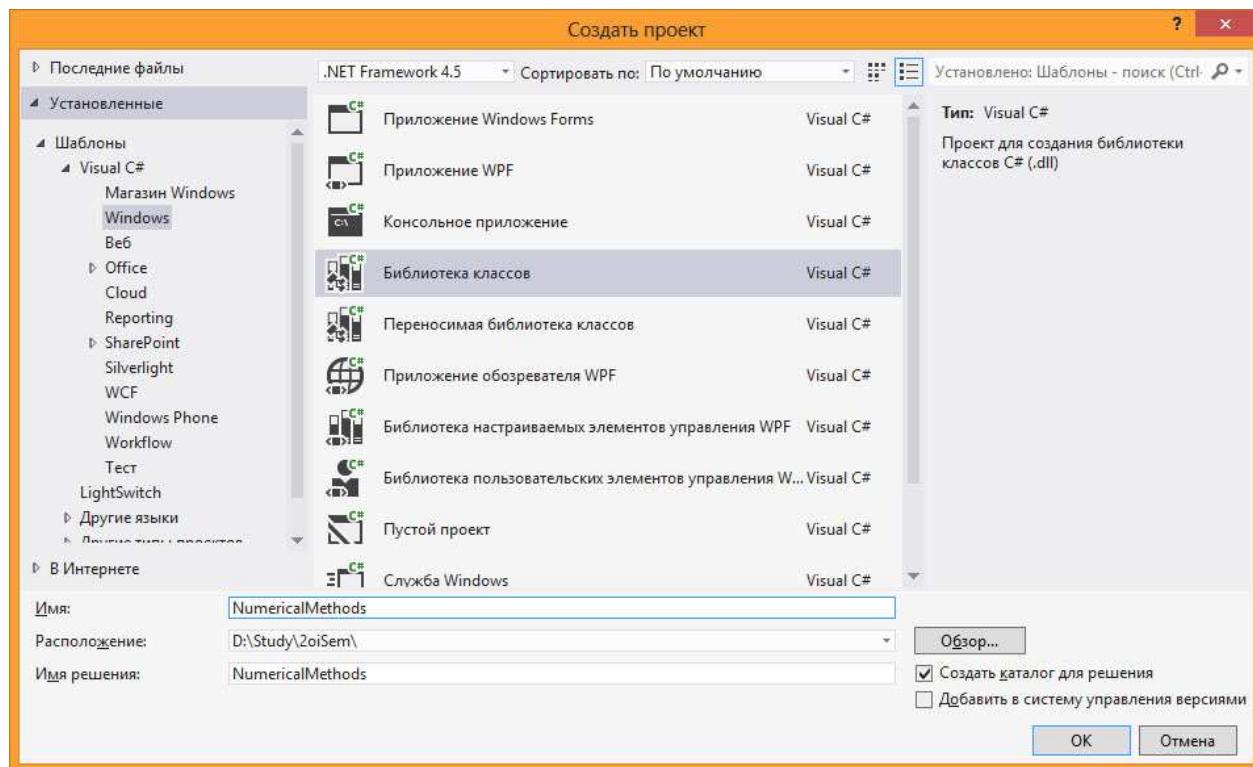


Рис. 6.6. Создание dll библиотеки

После вставки класса NumMeth необходимо проект скомпилировать. Это можно сделать через клавишу F5, но в этом случае после компиляции проекта появится ошибка, сообщающая о том, что dll библиотеку запустить самостоятельно не получится, а только в составе другого проекта. В любом случае в папке bin, которая находится в папке с проектом, появится dll библиотека, которую далее необходимо будет подключить к основному проекту. Чтобы избежать вывода ошибки, нужно нажать F6, в этом случае библиотека скомпилируется, но запускаться не станет.

Работу с классом NumMeth закончили. В следующем подразделе переходим к классу генерации матриц.

### Ссылка на весь исходный код класса:

[https://docs.google.com/document/d/1res0sVvbXb8Iq7wuRRDuZBjuE9TRhTqq4EQb73\\_qUUs](https://docs.google.com/document/d/1res0sVvbXb8Iq7wuRRDuZBjuE9TRhTqq4EQb73_qUUs) (перенос адреса)  
скопируйте весь адрес в строку браузера (перенос адреса)

## 6.3 Генерация матриц

В программе должны быть реализованы несколько способов ввода матриц: первый – вручную, второй и третий – автоматически. Второй, в свою очередь, должен быть реализован с помощью псевдослучайных величин, а третий – на основе десяти различных алгоритмов генерации специальных (плохо обусловленных) матриц.

При ручном вводе данные задаёт пользователь, автоматическая генерация, как на основе генератора псевдослучайных чисел, так и на основе специальных матриц, будут реализованы в одном классе. Этот класс реализуем в консольном проекте, после чего создадим из него dll библиотеку.

Прежде чем переходить к реализации, посмотрим, какие матрицы будем генерировать, то есть, определим, что в них общего и в чём различия (см. подразд. 3.6, с. 107). Все плохо обусловленные матрицы разделим на четыре типа:

1. Матрицы, генерируемые на основе порядка матрицы и номеров строк и столбцов (первая, четвертая и пятая матрицы).
2. Константные матрицы (третья, вторая и десятая матрицы).

3. Матрицы, генерируемые на основе аргумента, порядка, номеров строк и столбцов (седьмая, восьмая и девятая матрицы).
4. Матрица, генерируемая только на основе аргумента (шестая матрица).

Исходя из такой классификации, нам потребуются четыре различных конструктора для класса, в котором эти матрицы будут генерироваться. В каждом конструкторе одним из аргументов будет номер плохо обусловленной матрицы. Для матрицы, созданной на основе псевдослучайных чисел, определим номер 11. Остальные аргументы у разных конструкторов будут различны. Приступим к реализации.

Создали консольный проект. В нём создали новый класс SpecMatrix. Помимо конструкторов в классе будут реализованы методы для генерации всех матриц (пока пустые) и метод, который на основе значения переменной Type, будет решать, какой из методов генерации матрицы запускать (листинг 6.28).

### Листинг 6.28.

```
public class SpecMatrix {
    double[,] A;
    int N;
    int Type;
    double arg;
    public double[,] Matrix { get { return A; } }
    public SpecMatrix(int n, int t) {
        N = n;
        Type = t;
        genMatrix();
    }
    public SpecMatrix(int n, int t, double a) {
        N = n;
        Type = t;
        arg = a;
        genMatrix();
    }
    public SpecMatrix(double a) {
        Type = 6;
        arg = a;
        genMatrix();
    }
}
```

```
public SpecMatrix(int t) {  
    Type = t;  
    genMatrix();  
}  
Private void genMatrix() {  
    switch (Type) {  
        case 1:  
            Matrix1();  
            break;  
        case 2:  
            Matrix2();  
            break;  
        case 3:  
            Matrix3();  
            break;  
        case 4:  
            Matrix4();  
            break;  
        case 5:  
            Matrix5();  
            break;  
        case 6:  
            Matrix6();  
            break;  
        case 7:  
            Matrix7();  
            break;  
        case 8:  
            Matrix8();  
            break;  
        case 9:  
            Matrix9();  
            break;  
        case 10:  
            Matrix10();  
            break;  
        case 11:
```

```
    Matrix11();
    break;
}
}

private void Matrix1() {
}

private void Matrix2() {
}

private void Matrix3() {
}

private void Matrix4() {
}

private void Matrix5() {
}

private void Matrix6() {
}

private void Matrix7() {
}

private void Matrix8() {
}

private void Matrix9() {
}

private void Matrix10() {
}

private void Matrix11() {
}
```



Таким образом, сразу после создания экземпляра класса `SpecMatrix` на основе данных, которые были переданы в конструктор, будет сгенерирована матрица.

Перейдём непосредственно к генерации. Начнём с самого простого – с генерации на основе случайных чисел. В нашем случае для этого будет использоваться метод `Matrix11`.

**Листинг 6.29.**

```
private void Matrix11() {  
    {#  
        Random rnd = new Random();  
        A = new double[N, N];  
        for (int i = 0; i < N; i++) {  
            for (int j = 0; j < N; j++) {  
                A[i, j] = rnd.NextDouble() * 100 - 50;  
            }  
        }  
    #}  
}
```



Проверим вывод. Для этого в классе Program в методе Main создадим экземпляр класса SpecMatrix. В его конструктор передадим параметры, соответствующие генерации случайной матрицы.

**Листинг 6.30.**

```
static void Main(string[] args) {  
    {#  
        SpecMatrix matrix = new SpecMatrix(5, 11);  
        for (int i = 0; i < 5; i++) {  
            for (int j = 0; j < 5; j++) {  
                if (matrix.Matrix[i, j] >= 0) Console.Write(" ");  
                Console.Write(Math.Round(matrix.Matrix[i, j], 3));  
                Console.Write("\t");  
            }  
            Console.WriteLine();  
        }  
        Console.ReadKey();  
    #}  
}
```



Вывод правильный (рис. 6.7).

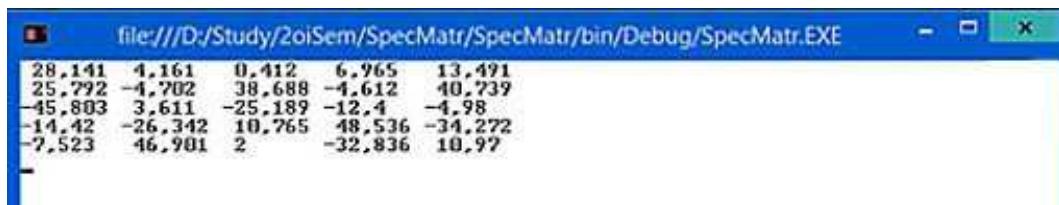


Рис. 6.7. Вывод случайной матрицы

Далее будем двигаться по порядку, с первой матрицы по десятую. Для первой матрицы (матрица Гильберта) элементы вычисляются по формуле:

$$a_{ij} = 1/(i + j - 1).$$

### Листинг 6.31.

```
private void Matrix1() {
{#
    A = new double[N, N];
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            A[i-1,j-1] = 1/((double)i+(double)j-1);
        }
    }
#}
}
```

Изменим создание экземпляра класса SpecMatrix:

```
SpecMatrix matrix = new SpecMatrix(5, 1);
```

И проверим вывод (рис. 6.8).

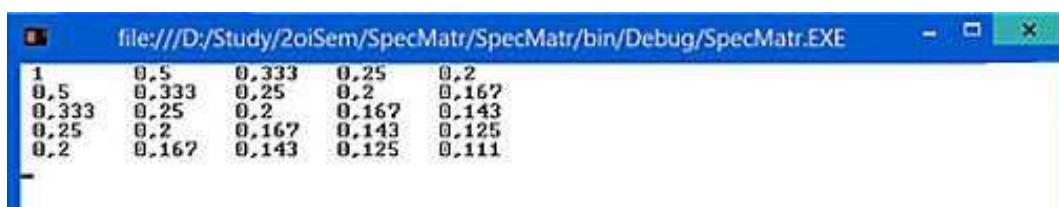


Рис. 6.8. Вывод матрицы первого типа

Размер второй матрицы известен. Он равен двадцати. Все элементы на главной диагонали равны единице, и все элементы  $a_{i,j}$ , где  $j = i + 1$ , также равны единице. Остальные элементы равны нулю.

### Листинг 6.32.

```
private void Matrix2() {  
    N = 20;  
    A = new double[N, N];  
    for (int i = 0; i < 20; i++) {  
        if (i != 19) {  
            A[i, i + 1] = 1;  
        }  
        A[i, i] = 1;  
    }  
}
```



Ниже (листинг 6.33) приведён код для метода Main.

### Листинг 6.33.

```
static void Main(string[] args) {  
    #  
    SpecMatrix matrix = new SpecMatrix(2);  
    #}  
    for (int i = 0; i < 20; i++) {  
        for (int j = 0; j < 20; j++) {  
            if (matrix.Matrix[i, j] >= 0) Console.Write(" ");  
            Console.WriteLine(Math.Round(matrix.Matrix[i, j], 3));  
        #  
        Console.WriteLine();  
    }  
    Console.ReadKey();  
}
```



Результат вычислений показан на рис. 6.9.

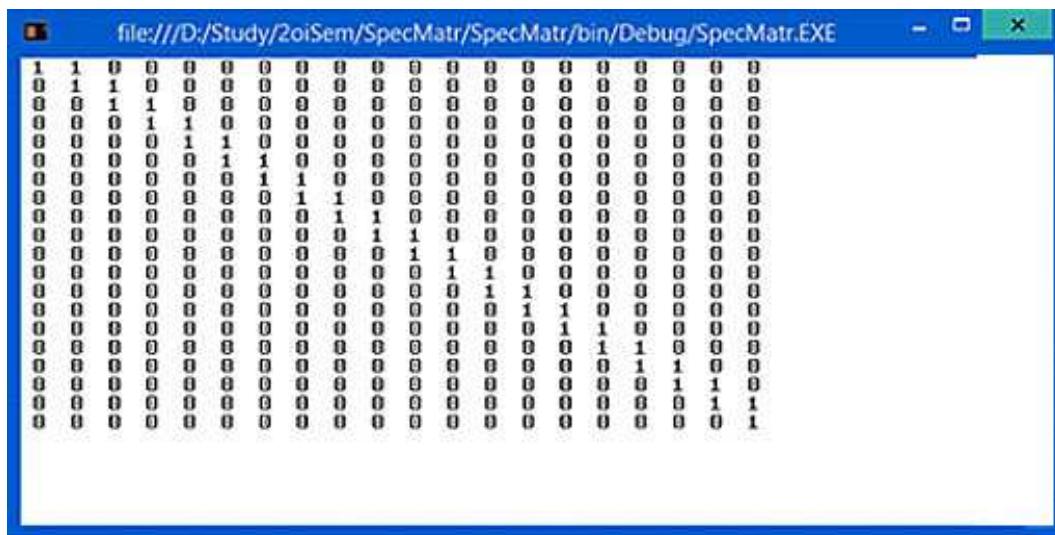


Рис. 6.9. Вывод матрицы второго типа

Третья матрица, как и вторая, фиксирована своим выражением:

$$A = \begin{bmatrix} 5 & 4 & 7 & 5 & 6 & 7 & 5 \\ 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 5 & 6 & 7 & 5 & 9 & 10 & 10 \end{bmatrix}$$

Эта матрица порождается кодом, показанным в листинге 6.34:

### Листинг 6.34.

```
private void Matrix3() {
{#
N = 7;
A = new double[,] {{5,4,7,5,6,7,5},
{4,12,8,7,8,8,6},
{7,8,10,9,8,7,7},
{5,7,9,11,9,7,5},
{6,8,8,9,10,8,9},
{7,8,7,7,8,10,10},
```

```
{5,6,7,5,9,10,10}};  
#}  
}
```

Формулы для вычисления элементов матриц четвёртого типа следующие:

$$a_{i,j} = \begin{cases} 0.01/[(n - i + 1)(i + 1)] & \text{при } i = j \\ 0 & \text{при } i < j \\ i(n - j) & \text{при } i > j \end{cases}$$

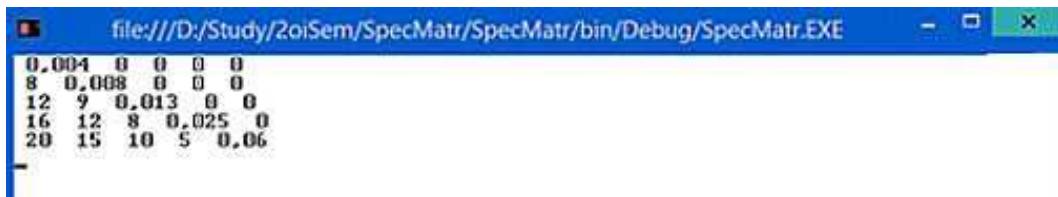


Рис. 6.10. Вывод матрицы четвёртого типа. Иной вариант: вместо операции умножения "\*" в седьмой строке листинга 6.35 здесь применена операция деления "/". Рекомендуем использовать основной вариант листинга 6.35

### Листинг 6.35.

```
private void Matrix4() {  
    #  
    A = new double[N, N];  
    for (int i = 1; i <= N; i++) {  
        for (int j = 1; j <= N; j++) {  
            if (i == j) {  
                A[i-1, j-1]=0.01/(((double)N-(double)i+1)*((double)i+1));  
            }  
            elseif (i < j) {  
                A[i - 1, j - 1] = 0;  
            }  
            else {  
                A[i-1,j-1] = i*(N-j);  
            }  
        }  
    }  
    #}  
}
```

Для проверки после вывода матрицы второго типа необходимо немного изменить код в классе Program метода Main:

### Листинг 6.36.

```
SpecMatrix matrix = new SpecMatrix{#(5,4);#
for (int i = 0; {#i < 5;#} i++) {
    for (int j = 0; {#j < 5;#} j++) {
        if (matrix.Matrix[i, j] >= 0) Console.WriteLine(" ");
        Console.WriteLine(Math.Round(matrix.Matrix[i, j],3));
        Console.WriteLine(" ");
    }
    Console.WriteLine();
}
Console.ReadKey();
```



Для вычисления значения элементов матрицы пятого типа используется следующая формула и соответствующий листинг 6.37:

$$a_{i,j} = \begin{cases} 0.01/[(n - i + 1)(i + 1)] & \text{при } i = j \\ j(n - i) & \text{при } i < j \\ i(n - j) & \text{при } i > j \end{cases}$$

### Листинг 6.37.

```
private void Matrix5() {
{#
A = new double[N, N];
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        if (i == j) {
            A[i-1, j-1]=0.01/(((double)N-(double)i+1)*((double)i+1));
        }
        else if (i < j) {
            A[i - 1, j - 1] = j*(N-i);
        }
        else {
            A[i - 1, j - 1] = i * (N - j);
        }
    }
}
```

```

    }
}
}
#}
}

```

Для вывода матрицы пятого типа также необходимо немного изменить код в методе Main класса Program:

### Листинг 6.38.

```

SpecMatrix matrix = new SpecMatrix{#(5,5);#}
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (matrix.Matrix[i, j] >= 0) Console.WriteLine(" ");
        Console.Write(Math.Round(matrix.Matrix[i, j],3));
        Console.WriteLine(" ");
    }
    Console.WriteLine();
}
Console.ReadKey();

```



Рис. 6.11. Вывод матрицы пятого типа. Иной вариант: вместо операции умножения “\*” в седьмой строке листинга 6.37 здесь применена операция деления “/”. Рекомендуем использовать основной вариант листинга 6.37. Подобное замечание см. для рис. 6.10

Элементы матрицы шестого типа задаются формулами (листинг 6.39):

$$A = \begin{bmatrix} R & S & T & T \\ S & R & S & T \\ T & S & R & S \\ T & T & S & R \end{bmatrix}, \quad R = \begin{bmatrix} \operatorname{ctg} \theta & \operatorname{cosec} \theta \\ -\operatorname{cosec} \theta & \operatorname{ctg} \theta \end{bmatrix},$$

$$S = \begin{bmatrix} 1 - \operatorname{ctg} \theta & \operatorname{cosec} \theta \\ -\operatorname{cosec} \theta & 1 + \operatorname{ctg} \theta \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Вычисления проводить при  $\theta$  близком к нулю или  $\pi$ .

### Листинг 6.39.

```
private void Matrix6() {
    {#
        N = 8;
        A = new double[N, N];
        double[,] T = new double[,] { { 1, 1 }, { 1, 1 } };
        double[,] R = new double[,] { { ctg(arg), cosec(arg) },
        { -cosec(arg), ctg(arg) } };
        double[,] S = new double[,] { { 1 - ctg(arg), cosec(arg) },
        { 1 - cosec(arg), 1 + ctg(arg) } };
        for (int i = 0; i < N; i += 2) {
            for (int j = 0; j < N; j += 2) {
                double[,] V = null;
                if (i == j) {
                    V = R;
                }
                else if (i == j + 2 || i + 2 == j) {
                    V = S;
                }
                else {
                    V = T;
                }
                for (int k = 0; k < 2; k++) {
                    for (int t = 0; t < 2; t++) {
                        Matrix[i + k, j + t] = V[k, t];
                    }
                }
            }
        }
    #}
}
```

В генерации матриц этого типа используются дополнительные функции (cosec, ctg), которые отсутствуют среди стандартных в библиотеке Math, поэтому необходимо их написать самостоятельно (листинг 6.40).

### Листинг 6.40.

```
double cosec(double arg) {  
    try {  
        return 1 / Math.Sin(arg);  
    }  
    catch (Exception e) {  
        return 0;  
    }  
}  
  
double ctg(double arg) {  
    try {  
        return Math.Sin(arg) / Math.Cos(arg);  
    }  
    catch (Exception e) {  
        return 0;  
    }  
}
```



Изменяем метод Main в классе Program (листинг 6.41):

### Листинг 6.41.

```
SpecMatrix matrix = new SpecMatrix{#(Math.PI);#}  
for (int i = 0; {#i < 8;#} i++) {  
    for (int j = 0; {#j < 8;#} j++) {  
        if (matrix.Matrix[i, j] >= 0) Console.WriteLine(" ");  
        Console.Write(Math.Round(matrix.Matrix[i, j], 3));  
        Console.WriteLine(" ");  
    }  
    Console.WriteLine();  
}  
Console.ReadKey();
```



и проверяем вывод матрицы шестого типа (рис. 6.12):

```

file:///D:/Study/ZoiSem/SpecMatr/SpecMatr/bin/Debug/SpecMatr.EXE
0 8,16588936419192E+15 1 8,16588936419192E+15 1 1 1
-8,16588936419192E+15 0 -8,16588936419192E+15 1 1 1
1 8,16588936419192E+15 0 8,16588936419192E+15 1 8,16588936419192E+15 1 1
-8,16588936419192E+15 1 -8,16588936419192E+15 0 -8,16588936419192E+15 1 1 1
1 1 1 8,16588936419192E+15 0 8,16588936419192E+15 1 8,16588936419192E+15
1 1 -8,16588936419192E+15 1 -8,16588936419192E+15 0 -8,16588936419192E+15 1
1 1 1 1 8,16588936419192E+15 0 8,16588936419192E+15
1 1 1 1 -8,16588936419192E+15 1 -8,16588936419192E+15 0

```

Рис. 6.12. Вывод матрицы шестого типа

Для вычисления значений элементов матрицы седьмого типа используем следующие формулы:

$$\begin{aligned} a_{ii} &= \alpha^{|n-2i|/2}; \\ a_{1j} &= a_{j1} = a_{11}/\alpha^j; \\ a_{nj} &= a_{jn} = a_{nn}/\alpha^j; \\ a_{ij} &= 0 \text{ для остальных значений } i \text{ и } j, \end{aligned}$$

где  $\alpha$  – параметр (аргумент функции), который в листинге 6.42 программы обозначаем arg:

### Листинг 6.42.

```

private void Matrix7() {
    #
    A = new double[N, N];
    for (int i = 1; i <= N; i++) {
        A[i-1, i-1]=Math.Pow(arg, Math.Abs((double)N-(double)i*2)/2);
    }
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            if (i != j) {
                if (i == 1 || j == 1) {
                    A[i-1, j-1]=Matrix[0, 0]/Math.Pow(arg, (double)j);
                }
                else if (i == N || j == N) {
                    A[i-1, j-1]=A[N-1, N-1]/Math.Pow(arg, (double)j);
                }
            }
        }
    }
}

```

```
    }
    else {
        A[i-1, j-1]=0;
    }
}
#
}
```

Вносим исправления метода Main (листинг 6.43):

## Листинг 6.43.

```
static void Main(string[] args) {
    SpecMatrix matrix = new SpecMatrix{#(5,7,2);#}
    for (int i = 0; {#i < 5;#} i++) {
        for (int j = 0; {#j < 5;#} j++) {
            if (matrix.Matrix[i, j] >= 0) Console.Write(" ");
            Console.Write(Math.Round(matrix.Matrix[i, j],3));
            Console.Write{#("\t");#}
        }
        Console.WriteLine();
    }
    Console.ReadKey();
}
```

Проверяем вывод матрицы седьмого типа (рис. 6.13):



Рис. 6.13. Вывод матрицы седьмого типа

Формула для вычисления значения элементов матрицы восьмого типа

$$a_{ij} = e^{i \cdot j \cdot h}$$

при  $h$  близких к нулю или 1000 реализована в листинге 6.44:

### Листинг 6.44.

```
private void Matrix8() {
{#
    A = new double[N, N];
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            A[i-1, j-1]=Math.Exp((double)i*(double)j*arg);
        }
    }
#}
}
```

Изменения метода Main включены в листинг 6.45, а проверка вывода матрицы восьмого типа показана на рис. 6.14:

### Листинг 6.45.

```
SpecMatrix matrix = new SpecMatrix{#(5,8,0.1);#}
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (matrix.Matrix[i, j] >= 0) Console.WriteLine(" ");
        Console.WriteLine(Math.Round(matrix.Matrix[i, j],3));
        Console.WriteLine("\t");
    }
    Console.WriteLine();
}
Console.ReadKey();
```

1.105	1.221	1.35	1.492	1.649
1.221	1.492	1.822	2.226	2.718
1.35	1.822	2.46	3.32	4.482
1.492	2.226	3.32	4.953	7.389
1.649	2.718	4.482	7.389	12.182

Рис. 6.14. Вывод матрицы восьмого типа

Для вычисления значения элементов матрицы девятого типа используется следующая формула:

$$a_{i,j} = c + \log_2(i \cdot j),$$

где  $c$  – аргумент, обозначаемый  $\text{arg}$  в следующем коде (листинг 6.46):

### Листинг 6.46.

```
private void Matrix9() {  
    #  
    A = new double[N, N];  
    for (int i = 1; i <= N; i++) {  
        for (int j = 1; j <= N; j++) {  
            A[i-1, j-1]=arg+Math.Log((double)i*(double)j, 2);  
        }  
    }  
    #}  
}
```



Изменения метода `Main` включены в листинг 6.47, а проверка вывода матрицы девятого типа показана на рис. 6.15:

### Листинг 6.47.

```
static void Main(string[] args) {  
    SpecMatrix matrix = new SpecMatrix(5,{#9,10#});  
    for (int i = 0; i < 5; i++) {  
        for (int j = 0; j < 5; j++) {  
            if (matrix.Matrix[i, j] >= 0) Console.Write(" ");  
            Console.WriteLine(Math.Round(matrix.Matrix[i, j],3));  
            Console.Write("\t");  
        }  
        Console.WriteLine();  
    }  
    Console.ReadKey();  
}
```



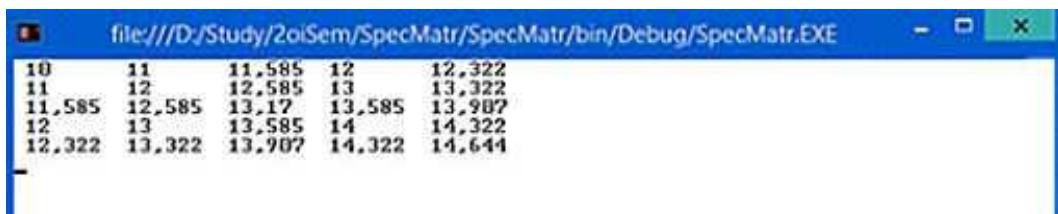


Рис. 6.15. Вывод матрицы девятого типа

Десятая матрица, также как вторая и третья, задана:

$$A = \begin{bmatrix} 0.9143 \cdot 10^{-4} & 0 & 0 & 0 \\ 0.8762 & 0.7156 \cdot 10^{-4} & 0 & 0 \\ 0.7943 & 0.8143 & 0.9504 \cdot 10^{-4} & 0 \\ 0.8017 & 0.6123 & 0.7165 & 0.7123 \cdot 10^{-4} \end{bmatrix}.$$

### Листинг 6.48.

```
private void Matrix10() {
{#
N = 4;
A = new double[,] {{0.9143*Math.Pow(10,-4),0,0,0},
{0.8762,0.756*Math.Pow(10,-4),0,0},
{0.794,0.8143,0.9504*Math.Pow(10,-4),0},
{0.8017,0.6123,0.7165,0.7123*Math.Pow(10,-4)} };
#}
}
```

Таким образом, генерация матриц закончена. По аналогии с примером из подразд. 6.2, из класса SpecMatrix нужно сделать dll библиотеку.

### Ссылка на исходный код класса:

[https://docs.google.com/document/d/1vkSAs1L2TeG11dbzpx6kVMvIb0mn1D\\_XY8JpxJSRWs/edit?usp=sharing](https://docs.google.com/document/d/1vkSAs1L2TeG11dbzpx6kVMvIb0mn1D_XY8JpxJSRWs/edit?usp=sharing)

(перенос адреса)  
(перенос адреса)  
скопируйте весь адрес  
в строку браузера

## 6.4 Создание пользовательского интерфейса и подключение ранее созданных библиотек

Перечислим формы и собственные элементы управления, которые будем использовать в создании пользовательского интерфейса:

1. Форма главного окна.
2. Форма ручного ввода матрицы А и вектора В для решения СЛАУ.
3. Форма ручного ввода матрицы для нахождения обратной матрицы.
4. Форма вывода результата.
5. Таблица для табулирования значений точности, времени выполнения и количества операций.
6. Панели для графиков.
7. Панель, помещённая на главную форму, для ввода параметров генерации матрицы или выбора ручного ввода.

Начнём с формы главного окна. На форме будут находиться `ListBox` (её пункты будут соответствовать задачам, которые программа решает:  $Ax = b$ ,  $A^{-1}$ ,  $\det A$ ), а также панель, записанная в п. 7. Создадим пока пустой класс `NumMetodLab` для этой панели, унаследованный от класса `FlowLayoutPanel`.

### Листинг 6.49.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace NumMetodLab {
    public class OptionMatrixPanel:FlowLayoutPanel {
    }
}
```



Свойства для формы:

- Text: Главное окно.
- WindowState: Maximized.

Кроме того, на саму форму нужно добавить контейнер FlowLayoutPanel со следующими свойствами:

- Location: 0,0.
- Anchor: Top, Bottom, Left, Right.
- Size: не имеет значения; главное, чтобы в редакторе форм панель FlowLayoutPanel была растянута по всей форме.

На FlowLayoutPanel добавим ListBox и растянем его по вертикали на всю доступную высоту.

Теперь создаём обработчик события, реагирующий на изменение размеров формы, в котором будем изменять высоту ListBox (рис. 6.16, а). Для этого выделим форму, на панели свойств перейдём во вкладку «События», найдём событие «Resize» и нажмём два раза на кнопке этого события, после чего откроется окно редактора кода с уже автоматически созданным методом.

В созданном методе необходимо написать следующий код:

### **Листинг 6.50.**

```
private void Form1_Resize(object sender, EventArgs e) {
    listBox1.Height = flowLayoutPanel1.Height;
}
```



Таким образом, при изменении размера окна будет изменяться и размер ListBox-а. Заполним ListBox нужными пунктами, а именно: «Решение СЛАУ», «Обращение матриц» и «Вычисление определителя». Для этого можно выделить ListBox в режиме конструктора форм, перейти на вкладку «Свойства», найти свойство «Items», нажать клавишу с многоточием, и в открывшемся окне записать нужные пункты: каждый пункт в новой строке (рис. 6.16, б).

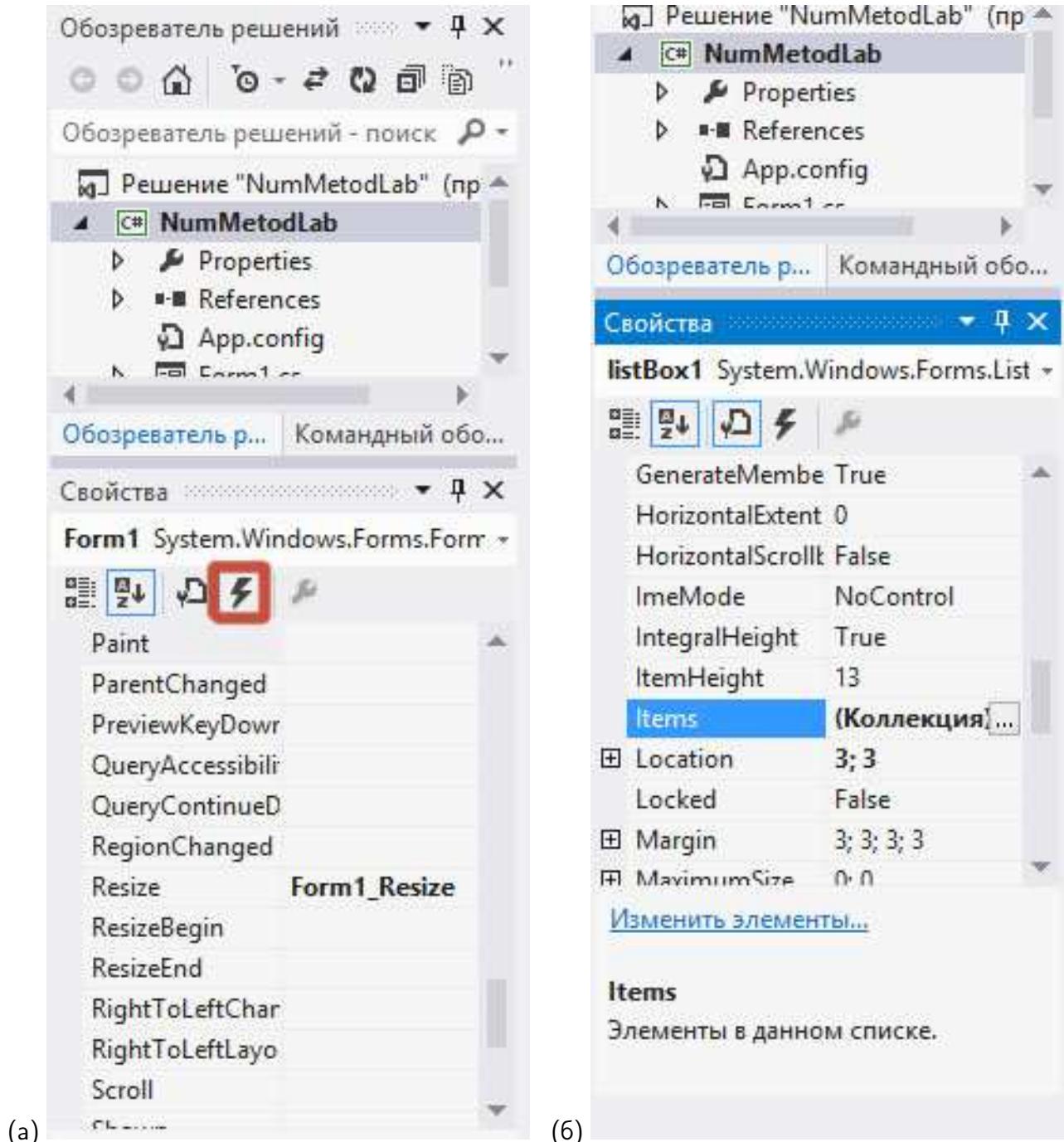


Рис. 6.16. Создание обработчика событий (а). Изменение элементов в ListBox (б)

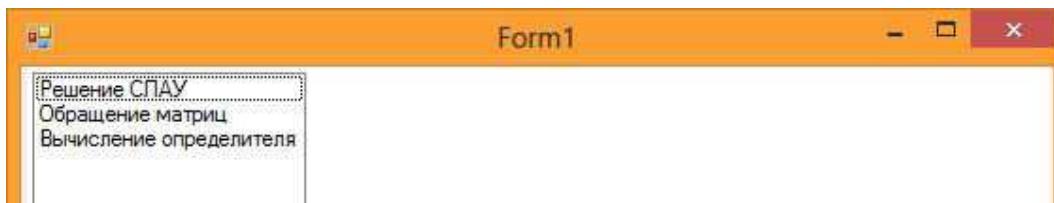


Рис. 6.17. Главная форма приложения

В результате после запуска приложения должна появиться главная форма приложения, похожая на форму, представленную на рис. 6.17.

Вернёмся теперь к классу, который будет представлять собой панель для ввода параметров матрицы.

В первую очередь создадим для него конструктор. Поскольку эта панель будет использоваться для ввода матриц при выполнении операций обращения, поиска решения СЛАУ или вычисления определителя, необходимо в качестве одного из аргументов передать в конструктор число, символизирующее то, с чем этот класс будет иметь дело. Это нужно для правильного отображения заголовка и для открытия нужной формы ввода, если будет выбран ручной ввод в раскрывающемся списке, который тоже скоро добавим. Кроме числа, нужно передать размеры панели, которые тоже будут изменяться вместе с изменением размеров главного окна приложения (см. рис. 6.17).

Следующий код необходимо дописать в тело класса OptionMatrixPanel:

### Листинг 6.51.

```
int type; // Тип страницы.
FlowLayoutPanel[] panels = new FlowLayoutPanel[5];
// Панели, на которые будут добавляться компоненты.
ComboBox[] comb = new ComboBox[2];
// Выпадающие списки для определения способа ввода и типа
// специальных матриц.
TextBox int_ot_txt = new TextBox(); // Текстовое поле для ввода
// начала интервала, с которого будет изменяться порядок матрицы.
TextBox int_do_txt = new TextBox(); // Текстовое поле для ввода
// конца интервала, до которого будет изменяться порядок матрицы.
TextBox step_txt = new TextBox(); // Текстовое поле для ввода шага,
// с которым будет изменяться порядок матрицы.
TextBox arg = new TextBox(); // Текстовое поле для ввода аргумента,
// который необходим для генерации некоторых матриц.
```

```
Button btn_go = new Button(); //Кнопка, при нажатии на которую,  
// будут происходить какие-то операции (ввод матриц вручную,  
// решение СЛАУ, обращение матриц, вычисление определителя).  
Label title = new Label(); //Заголовок страницы.  
TextBox matrix_input = new TextBox(); //Текстовое поле для ввода  
// матрицы, для вычисления определителя.  
public OptionMatrixPanel(int width, int height, int t) {  
    this.AutoSize = false;  
    type = t;  
    switch (type) {  
        case 1:  
            title.Text = "Решение СЛАУ";  
            break;  
        case 2:  
            title.Text = "Обращение матриц";  
            break;  
        case 3:  
            title.Text = "Вычисление определителя";  
            break;  
    }  
    this.Controls.Add(title);  
    Resize(width, height);  
    if (type == 3) {  
        InitComponents2();  
        return;  
    }  
    InitComponents();  
}  
  
private void InitComponents2() {}  
private void InitComponents() {}  
  
public void Resize(int width,int height) {  
    this.AutoSize = false;  
    this.Size = new System.Drawing.Size(width, height);  
    title.Width = this.Width;  
    for (int i = 0; i < 5; i++) {
```

```

if(panels[i] !=null)
    panels[i].Width = this.Width;
}
}

```



Как уже было сказано в предыдущем подразделе (см. подразд. 6.3 на с. 248), матрицы могут быть сгенерированы на основе порядка матрицы, аргумента или могут быть заданы по умолчанию (конечными формулами). Предлагаем для простоты внутри класса OptionMatrixPanel реализовать пять дополнительных панелей:

1. Панель с полями ввода интервала и шага для изменения периода матрицы.
2. Панель с полем для ввода аргумента.
3. Панель с выпадающим списком, в котором можно будет выбрать тип специальной матрицы.
4. Панель с кнопкой.
5. Панель с выпадающим списком, в котором можно будет выбрать тип ввода матрицы.

Вынести все эти элементы на отдельные панели желательно для того, чтобы для каждого ввода или типа матрицы не создавать их заново, а делать видимыми или невидимыми. Из конструктора класса видно, что страница озаглавливается в зависимости от выбора операции, которая будет выполняться (обращение матриц, решение СЛАУ или вычисление определителя). Для вычисления определителя матриц сформируем отдельный интерфейс.

Сначала заполним содержимое метода `InitComponents`, который будет вызываться для формирования пользовательского интерфейса при выборе обращения матриц или решения СЛАУ.

### **Листинг 6.52.**

```

private void InitComponents() {
    String[] combo_list = new String[] {"Вручную", "Случайные
                                         матрицы", "Спецматрицы"};
    Label zag = new Label();
    for (int i = 0; i < 5; i++) {

```

```
panels[i] = new FlowLayoutPanel();
panels[i].AutoSize = false;
panels[i].Height = 28;
panels[i].Width = this.Width;
panels[i].AutoScroll = true;
this.Controls.Add(panels[i]);
}

zag.Text = "Выберите тип ввода матрицы:";
zag.AutoSize = true;
zag.Margin = new System.Windows.Forms.Padding(0, 6, 0, 0);
comb[0] = new ComboBox();
comb[0].DropDownStyle = ComboBoxStyle.DropDownList;
comb[0].Items.AddRange(combo_list);
comb[0].SelectedIndex = 0;
panels[0].Controls.Add(zag);
panels[0].Controls.Add(comb[0]);
zag = new Label();
zag.Text = "Выберите тип матрицы: ";
zag.Margin = new System.Windows.Forms.Padding(0, 6, 0, 0);
zag.AutoSize = true;
comb[1] = new ComboBox();
for (int i = 1; i <= 10; i++) {
    comb[1].Items.Add(i.ToString());
}
comb[1].DropDownStyle = ComboBoxStyle.DropDownList;
comb[1].SelectedIndex = 0;
panels[1].Controls.Add(zag);
panels[1].Controls.Add(comb[1]);
zag = new Label();
zag.Text = "Изменение порядка матрицы, от: ";
zag.Margin = new System.Windows.Forms.Padding(0, 6, 0, 0);
zag.AutoSize = true;
int_ot_txt.Text = "5";
panels[2].Controls.Add(zag);
panels[2].Controls.Add(int_ot_txt);
int_ot_txt.GotFocus += int_txt_GotFocus;
int_ot_txt.LostFocus += int_txt_LostFocus;
```

```
zag = new Label();
zag.Text = " "; до: ";
zag.Margin = new System.Windows.Forms.Padding(0, 6, 0, 0);
zag.AutoSize = true;
int_do_txt.Text = "100";
panels[2].Controls.Add(zag);
panels[2].Controls.Add(int_do_txt);
int_do_txt.GotFocus += int_txt_GotFocus;
int_do_txt.LostFocus += int_txt_LostFocus;
zag = new Label();
zag.Text = " "; шаг: ";
zag.Margin = new System.Windows.Forms.Padding(0, 6, 0, 0);
zag.AutoSize = true;
step_txt.Text = "5";
panels[2].Controls.Add(zag);
panels[2].Controls.Add(step_txt);
step_txt.GotFocus += int_txt_GotFocus;
step_txt.LostFocus += int_txt_LostFocus;
zag = new Label();
zag.Text = "Значение аргумента: ";
zag.Margin = new System.Windows.Forms.Padding(0, 6, 0, 0);
zag.AutoSize = true;
arg.Text = "1";
arg.GotFocus += int_txt_GotFocus;
arg.LostFocus += arg_LostFocus;
panels[3].Controls.Add(zag);
panels[3].Controls.Add(arg);
btn_go.Text = "Создать отчёт";
btn_go.AutoSize = true;
panels[4].Controls.Add(btn_go);
panels[4].Height = 30;
}

void arg_LostFocus(object sender, EventArgs e) {
    string arr = "0123456789";
    string msg = ((TextBox)sender).Text.Trim();
    try {
```

```
double a = Convert.ToDouble(msg.Replace(".", ",",""));
((TextBox)sender).Text = msg.Replace(".", ",","");
}
catch (Exception e3) {
    ((TextBox)sender).Text = ((TextBox)sender).Name;
    MessageBox.Show("В это поле может быть введено только
вещественное число!");
}
}

void int_txt_LostFocus(object sender, EventArgs e) {
string arr = "0123456789";
string msg = ((TextBox)sender).Text.Trim();
for (int i = 0; i < arr.Length; i++) {
    msg = msg.Replace(arr[i].ToString(), "");
}
if (msg != "") {
    ((TextBox)sender).Text = ((TextBox)sender).Name;
    MessageBox.Show("В это поле может быть введено только целое
положительное число!");
}
}

void int_txt_GotFocus(object sender, EventArgs e) {
    ((TextBox)sender).Name = ((TextBox)sender).Text;
}
```



Для всех текстовых полей необходимо сделать проверку на введённые данные. В текстовые поля для определения размера матрицы можно будет вводить только целые числа, в текстовое поле с аргументом можно будет вводить любое вещественное число. В первую очередь, необходимо обработать событие получения фокуса текстовыми полями и запись в переменную `Name`, которая доступна как для чтения, так и для записи из всех классов, унаследованных от класса `Component`, значения которого и находятся в этом текстовом поле (можно было бы выделить для этого отдельную переменную, но переменная `Name` нигде больше использоваться не будет, т. е., такое решение оправдано).

После потери фокуса для полей, в которых должно находиться целое положительное число, будет вызван метод `int_txt_LostFocus`, в котором будет проверено содержание поля на присутствие недопустимых символов. Если такие символы будут найдены, то текст в этом поле примет значение, которое было в нём до получения фокуса, а это значение сохранено в переменной `Name`. При потере фокуса текстовым полем, в которое можно ввести только вещественные числа, будет вызван метод `arg_LostFocus`, в котором произведётся попытка перевода текста в вещественное число. Если этого сделать не получится (например, в поле были введены недопустимые символы), то вызовется исключение, которое будет обработано в блоке `catch`. Текст вернётся в своё первоначальное состояние (до получения фокуса полем), и появится сообщение об ошибке ввода.

Таким образом, панели созданы. Теперь необходимо организовать их «видимость» в зависимости от выбранных пунктов в выпадающих списках. По умолчанию ввод осуществляется вручную. Значит, должны быть видны только панель с первым выпадающим списком и панель с кнопкой. Для этого сразу после вызова метода `InitComponents`, в конструкторе класса запишем следующее:

### Листинг 6.53.

```
InitComponents();
{#
panels[1].Visible = false;
panels[2].Visible = false;
panels[3].Visible = false;
#}
```

Назначим обработчики событий для выбора элемента в выпадающих списках. Для этого в конце метода `InitComponents` допишем следующее:

### Листинг 6.54.

```
comb[0].SelectedIndexChanged += combo1_SelectedIndexChanged;
comb[1].SelectedIndexChanged += combo2_SelectedIndexChanged;
```

Внутри класса создадим два соответствующих метода:

**Листинг 6.55.**

```
void combo2_SelectedIndexChanged(object sender, EventArgs e) {  
}  
void combo1_SelectedIndexChanged(object sender, EventArgs e) {  
}
```



Сначала напишем содержимое метода `combo1_SelectedIndexChanged`. Если выбран первый элемент, то видимыми будут нулевая и четвёртая панели (ручной ввод). Если второй, то видимыми будут нулевая, вторая и четвёртая панели (генерация случайных матриц). Если будет выбран третий элемент, то видимыми должны быть нулевая, первая и четвёртая панели, а также могут быть видимы и другие панели, в зависимости от того, какая матрица выбрана во втором выпадающем списке. Для этого создадим ещё один метод, весь код которого будет записан в методе `combo1_SelectedIndexChanged`. Код этого метода приведён ниже (вместе с пока пустым методом «заглушкой», в котором будут показываться необходимые панели для указания параметров генерации специальных матриц):

**Листинг 6.56.**

```
void combo1_SelectedIndexChanged(object sender, EventArgs e) {  
    switch (comb[0].SelectedIndex) {  
        case 0: {  
            for (int i = 1; i < 4; i++)  
                panels[i].Visible = false;  
        }  
        break;  
        case 1: {  
            for (int i = 1; i < 4; i++)  
                panels[i].Visible = false;  
            panels[2].Visible = true;  
        }  
        break;  
        case 2: {  
            for (int i = 1; i < 4; i++)  
                panels[i].Visible = false;  
            SpecMatrixSelect();  
        }  
    }  
}
```

```
    }
    break;
}
}

void SpecMatrixSelect() {
}
```



Следующим шагом будет заполнение метода SpecMatrixSelect. Для этого необходимо вспомнить способы генерации матриц. Это было описано в предыдущем разделе (см. подразд. 6.3, начинающийся на с. 248), поэтому сразу приведём готовую реализацию метода SpecMatrixSelect:

### Листинг 6.57.

```
void SpecMatrixSelect() {
    panels[1].Visible = true;
    switch (comb[1].SelectedIndex+1) {
        case 1:
            panels[2].Visible = true;
            break;
        case 4:
            panels[2].Visible = true;
            break;
        case 5:
            panels[2].Visible = true;
            break;
        case 6:
            panels[3].Visible = true;
            break;
        case 7: {
            panels[3].Visible = true;
            panels[2].Visible = true;
        }
            break;
        case 8: {
            panels[2].Visible = true;
            panels[3].Visible = true;
        }
    }
}
```

```
        }
        break;
    case 9: {
        panels[2].Visible = true;
        panels[3].Visible = true;
    }
    break;
}
}
```



Теперь скроем все ненужные панели и вызовем этот метод из метода combo2\_SelectedIndexChanged:

### Листинг 6.58.

```
void combo2_SelectedIndexChanged(object sender, EventArgs e)
{
{#
for (int i = 1; i < 4; i++) panels[i].Visible = false;
SpecMatrixSelect();
#}
}
```



Далее добавим эту панель на главную форму, а точнее, три таких панели: две невидимые и одну видимую. Так как по умолчанию будет выбран пункт «Решение СЛАУ», то видимой по умолчанию будет первая панель. Для добавления вернёмся в класс главной формы. Сначала внутри класса создадим массив из только что созданных панелей, который будет состоять из трёх элементов:

### Листинг 6.59.

```
public partial classForm1 : Form {
{#
OptionMatrixPanel[] option_panel = new OptionMatrixPanel[3];
#}
```



В конструкторе класса после вызова метода `InitializeComponent` объявим элементы этого массива, добавим их на форму и сделаем два из них невидимыми. Добавим обработчик события выбора элемента в `ListBox`-е. В этом обработчике будем делать нужные панели видимыми, а ненужные – невидимыми:

### Листинг 6.60.

```
public Form1() {
    InitializeComponent();
    {#
    for (int i = 0; i < 3; i++) {
        option_panel[i] = new OptionMatrixPanel(flowLayoutPanel1.Width -
            listBox1.Width - 10, flowLayoutPanel1.Height, i+1);
        flowLayoutPanel1.Controls.Add(option_panel[i]);
        if (i > 0) option_panel[i].Visible = false;
    }
    listBox1.SelectedIndex = 0;
    listBox1.SelectedIndexChanged += listBox1_SelectedIndexChanged;
    #}
}

{#
void listBox1_SelectedIndexChanged(object sender, EventArgs e) {
    for (int i = 0; i < 3; i++) option_panel[i].Visible = false;
    option_panel[listBox1.SelectedIndex].Visible = true;
}
#}
```

Теперь пришло время протестировать написанную панель (рис. 6.18 и рис. 6.19).

Всё выводится, как и было запланировано. Однако пока ничего не выводится при выборе пункта «Вычисление определителя». Это было отложено и сейчас это исправим. Вернёмся к классу `OptionMatrixPanel` и в методе `InitComponents2` допишем следующее:

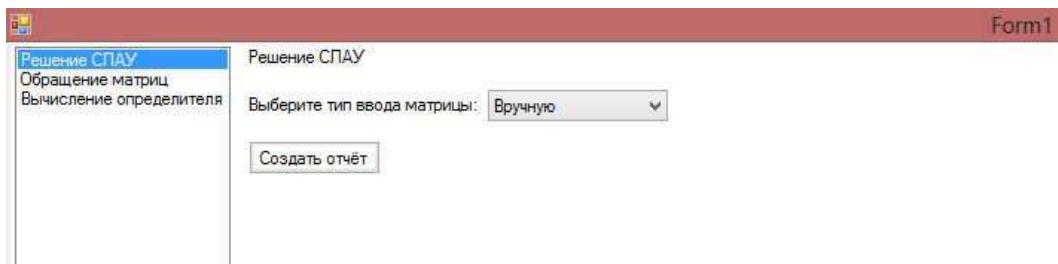


Рис. 6.18. Скриншот главной формы приложения (ввод данных для решения СЛАУ)

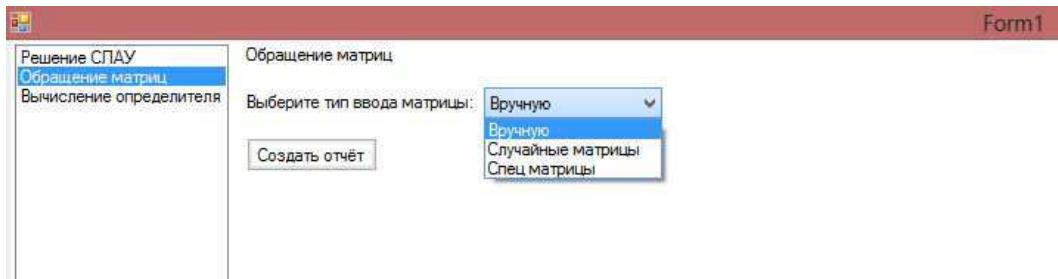


Рис. 6.19. Скриншот главной формы приложения (ввод данных для обращения матрицы)

### Листинг 6.61.

```
private void InitComponents2() {
    for (int i = 0; i < 3; i++) {
        panels[i] = new FlowLayoutPanel();
        panels[i].AutoSize = false;
        panels[i].Width = this.Width;
        panels[i].AutoScroll = true;
        this.Controls.Add(panels[i]);
    }
    Label zag = new Label();
    panels[0].Height = 28;
    zag.Text = "Введите матрицу для вычисления определителя (элементы в строке разделяйте пробелом, каждая строка матрицы вводится с новой строчки)";
    panels[0].Controls.Add(zag);
    zag.AutoSize = true;
    matrix_input.Multiline = true;
    matrix_input.Size = new System.Drawing.Size(200, 200);
    panels[1].Controls.Add(matrix_input);
    panels[1].Height = 220;
```

```

btn_go.Text = "Вычислить определитель";
btn_go.AutoSize = true;
panels[2].Controls.Add(btn_go);
}

```

В результате проверяем, что главная форма приложения с вводом данных для вычисления определителя появляется, как запланировано (рис. 6.20).

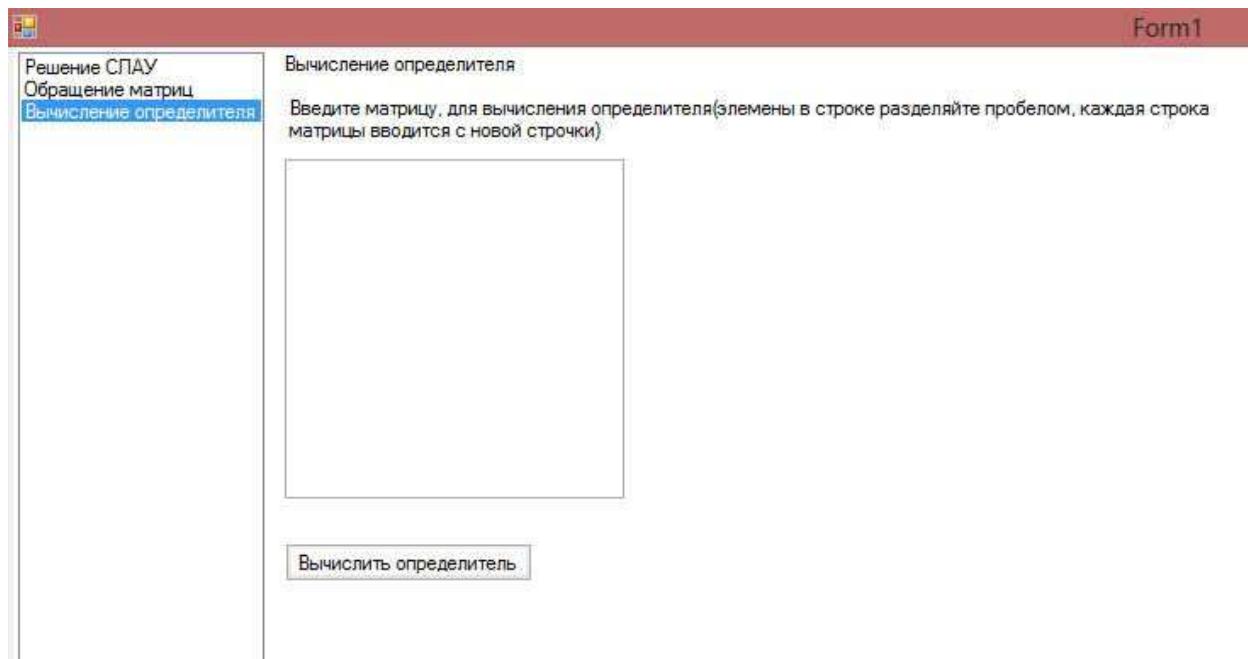


Рис. 6.20. Скриншот главной формы приложения (ввод данных для вычисления определителя)

С главной формой закончили. Перейдём к формам, через которые будут вводиться матрицы вручную для их обращения и для решения СЛАУ. Создадим новую форму в проекте. Для этого необходимо нажать правой кнопкой мыши на проекте, выбрать пункт «Добавить», далее выбрать пункт «форма Windows...». Вводим название формы и нажимаем «Добавить».

На форме (рис. 6.21) должны находиться три текстовых поля, свойство `Multiline` в `true` и две кнопки:

- `Textbox1` – поле под матрицу A.
- `Textbox2` – поле под вектор X установлено.

- Textbox3 – поле под вектор B.
- Button1 – кнопка «Принять».
- Button2 – кнопка «Отмена».

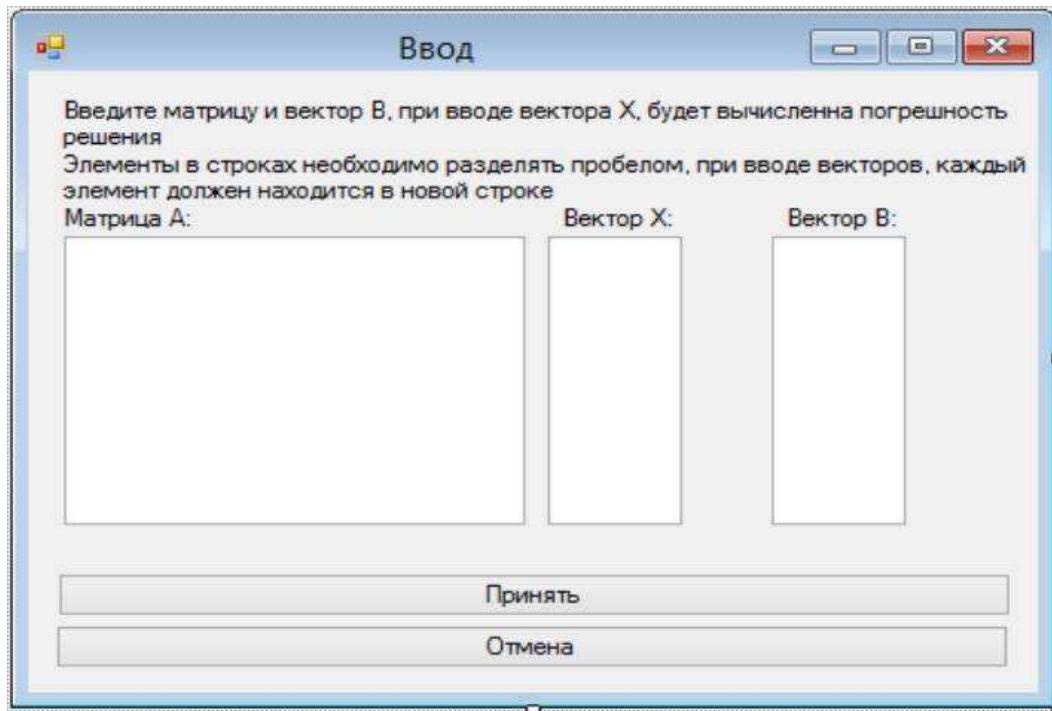


Рис. 6.21. Внешний вид формы HandInputSLAU

В свойство CancelButton устанавливаем значение «button2».

С интерфейсом разобрались. Переходим к написанию кода. Основной код будет находиться в обработчике события нажатия мыши на кнопке «Принять». Для того чтобы автоматически сгенерировать обработчик этого события в коде, необходимо в режиме конструктора два раза нажать на кнопку «Принять». Редактор автоматически перейдёт в режим написания кода и создаст метод, обрабатывающий событие:

### Листинг 6.62.

```
private void button1_Click(object sender, EventArgs e) {  
}
```

В самом классе необходимо создать 3 поля, доступные для чтения: первое поле – это двумерный массив (матрица A), второе и третье поля – одномерные массивы, доступные также для чтения (векторы X и B, соответственно):

### Листинг 6.63.

```
double[,] a;
double[] b;
double[] x;
public double[,] A { get { return a; } }
public double[] X { get { return x; } }
public double[] B { get { return b; } }
```



Теперь перейдём непосредственно к обработке текста и формированию из него массивов. Весь код для этого будет находиться в методе button1\_Click.

### Листинг 6.64.

```
private void button1_Click(object sender, EventArgs e) {
    try {
        string[] rowsA = textBox1.Text.Split(new char[] { '\r' });
        int n = rowsA.Length;
        a = new double[n, n];
        for (int i = 0; i < n; i++) {
            rowsA[i] = rowsA[i].Replace("\n", "");
            string[] cols = rowsA[i].Trim().Split(new char[] { ' ' });
            if (cols.Length != n) {
                MessageBox.Show("Данные введены некорректно!");
                return;
            }
            for (int j = 0; j < n; j++) {
                a[i, j] = Convert.ToDouble(cols[j].Replace(".", ","));
            }
        }
        string[] rowsB = textBox3.Text.Split(new char[] { '\r' });
        if (rowsB.Length != n) {
            MessageBox.Show("Данные введены некорректно!");
            return;
        }
    }
```

```
b = new double[n];
for (int i = 0; i < n; i++) {
    rowsB[i] = rowsB[i].Replace("\n", "").Trim();
    b[i] = Convert.ToDouble(rowsB[i].Replace(".", ","));
}
if (textBox2.Text.Trim() != "") {
    string[] rowsX = textBox2.Text.Split(new char[] {'\r'});
    if (rowsX.Length == n) {
        x = new double[n];
        for (int i = 0; i < n; i++) {
            rowsX[i] = rowsX[i].Replace("\n", "").Trim();
            x[i] = Convert.ToDouble(rowsX[i].Replace(".", ",")));
        }
    }
} else {
    MessageBox.Show("Вектор X был введён неправильно!
(поле можно оставить пустым)");
    return;
}
this.DialogResult = System.Windows.Forms.DialogResult.OK;
this.Close();
}
catch (Exception exception) {
    MessageBox.Show("Неверный формат введённых данных!");
}
```



В методе `button1_Click` выполняются попытки сформировать массивы на основе данных, введённых пользователем в соответствующие окна. Если возникнут исключения, то пользователь будет об этом проинформирован. Если же данные окажутся корректными, то будут сформированы массивы. Форма закроет и вернёт в класс, из которого она вызывалась, значение `OK`, что будет свидетельствовать об успешном создании массивов. Таким образом, эту форму необходимо будет вызывать в диалоговом режиме.

Создадим сразу вторую форму ввода матриц, для которых необходимо будет найти обратную. Назовём ее «HandInputINV» (рис. 6.22).

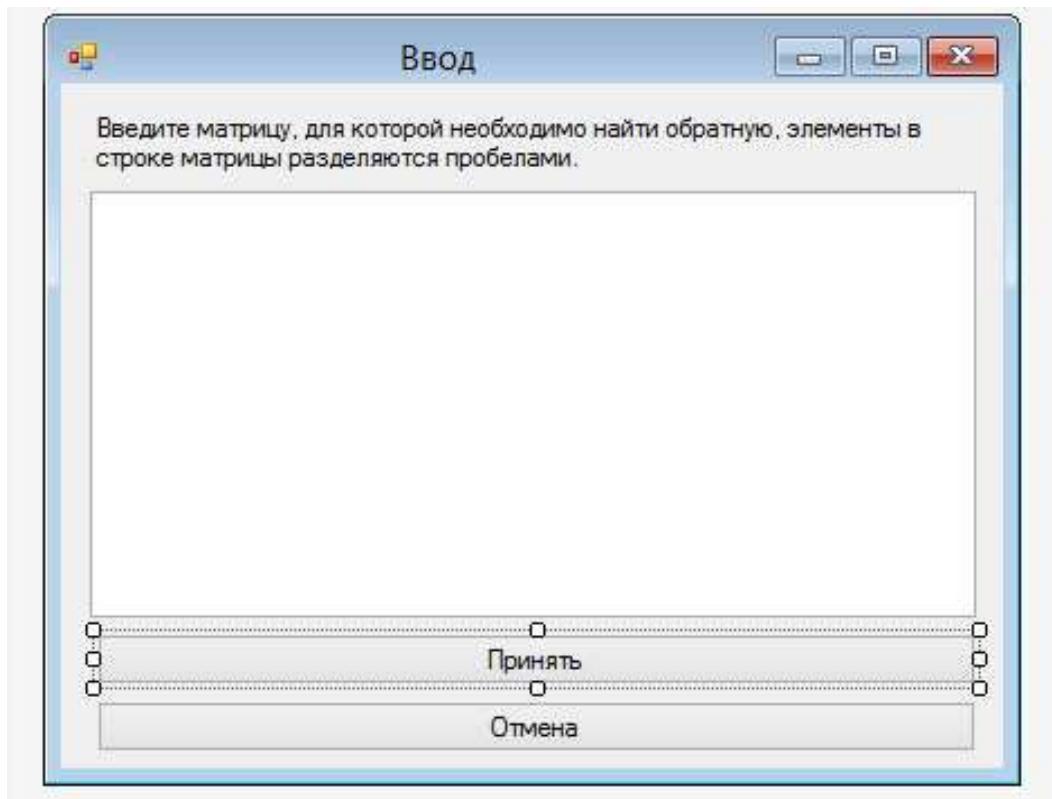


Рис. 6.22. Внешний вид формы HandInputINV

Формат ввода матрицы, свойства главной формы и текстового поля аналогичны форме HandInputSLAU. Исходный код класса представлен ниже:

### Листинг 6.65.

```
public partial class HandInputINV : Form {
    double[,] a = null;
    public double[,] A { get { return a; } }
    public HandInputINV() {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e) {
        try {
            string[] rowsA = textBox1.Text.Split(new char[] { '\r' });
            int n = rowsA.Length;
```

```
a = new double[n, n];
for (int i = 0; i < n; i++) {
    rowsA[i] = rowsA[i].Replace("\n", "");
    string[] cols = rowsA[i].Trim().Split(new char[] { ' ', ',' });
    if (cols.Length != n) {
        MessageBox.Show("Данные введены некорректно!");
        return;
    }
    for (int j = 0; j < n; j++) {
        a[i, j] = Convert.ToDouble(cols[j].Replace(".", ","));
    }
}
this.DialogResult = System.Windows.Forms.DialogResult.OK;
this.Close();
}
catch (Exception excetion) {
    MessageBox.Show("Неверный формат введённых данных!");
}
}
}
}
```



Теперь создадим класс формы для вывода отчёта, а также класс, который будет формировать и отображать таблицу, и класс, в котором будут строиться графики.

Сначала необходимо создать форму и добавить на неё контейнер, названный `FlowLayoutPanel`; его размеры надо сделать такими же, как и у формы; модификатор доступа установить в `public`; свойству `AutoScroll` установить значение `true`. Мы назвали эту форму «`FormOutput`».

Теперь необходимо создать класс для построения и отображения таблицы. Предлагаем следующий подход: у класса будет два конструктора, в первый будут передаваться 5 массивов (для отчёта по решению СЛАУ), во второй будут передаваться 8 массивов (для отчёта по обращению матриц) «`TablePanel`».

Исходный код приведён ниже:

### Листинг 6.66.

```
// Класс для создания таблицы
```

```

class TablePanel : TableLayoutPanel {
// Унаследован от контейнера TableLayoutPanel.
    double eps;
// В значение этой переменной будет записан машинный эпсилон.
// Массив из двух цветов,
// чтобы проще было читать данные из таблицы.
    System.Drawing.Color[] colors = new System.Drawing.Color[]
    { System.Drawing.Color.White, System.Drawing.Color.LightGray };
// Конструктор:
// В этот конструктор передается 5 массивов, в теле
// генерируется таблица для отчёта по решению СЛАУ, array1 -
// массив с порядками матриц, array2 - массив с временем
// выполнения операций, array3 - массив с погрешностями,
// array4 - массив с теоретическим числом операций, array5 -
// массив с фактическим числом операций.
    public MyTable(int[] array1, double[] array2, double[] array3,
double[] array4, double[] array5) {
        double a = 1; // Вычисляем машинный эпсилон.
        do { eps = a; a /= 2; } while (a != 0);
        this.Padding = new Padding(0, 0, 0, 0);
// Обнуляем отступы в таблице.
        this.RowCount = array1.Length + 1;
// Вычисляем и присваиваем количество строк.
        this.ColumnCount = 5; // Присваиваем количество столбцов.
        this.AutoSize = true;
// Устанавливаем автоматический подгон размеров.
// Строковый массив с заголовками столбцов.
        string[] titles = new string[] { "Порядок", "Время",
"Точность", "Теор. ЧО", "Факт. ЧО"};
        for (int i = 0; i < titles.Length; i++) {
// В этом цикле создаются заголовки столбцов
        Label l = new Label();
// Создание метки с текстом (заголовок столбца).
        l.Text = titles[i]; // Присваиваем название столбца.
        l.AutoSize = false; // Устанавливаем точные размеры,
        l.Width = 125; // иначе таблица может получиться
                      // "кривой".

```

```
l.Height = 20;
l.Margin = new System.Windows.Forms.Padding(0, 0, 0, 0);
// Убираем отступы.
l.BackColor = System.Drawing.Color.DarkGray;
// Закрашиваем в темно-серый цвет.
l.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
// Выравнивание текста по центру.
l.ForeColor = System.Drawing.Color.White;
// Цвет текста - белый.
this.Controls.Add(l);
// Добавляем элемент в таблицу.
}
for (int i = 0; i < array1.Length; i++) {
// Заполняем таблицу данными.
Label[] labs = new Label[5];
// В каждой строке будет 5 столбцов.
for (int j = 0; j < 5; j++) { // Создаём и оформляем ячейки.
    labs[j] = new Label();
    labs[j].Margin = new
        System.Windows.Forms.Padding(0, 0, 0, 0);
    labs[j].AutoSize = false;
    labs[j]. TextAlign =
        System.Drawing.ContentAlignment.MiddleCenter;
    labs[j].BorderStyle =
        System.Windows.Forms.BorderStyle.FixedSingle;
    labs[j].BackColor = colors[i % 2];
    labs[j].Width = 125;
    labs[j].Height = 20;
    this.Controls.Add(labs[j]);
}
labs[0].Text = array1[i].ToString();
// Заполнение ячеек данными.
labs[1].Text = Mant(array2[i]);
labs[2].Text = Mant(array3[i]);
labs[3].Text = Mant(array4[i]);
labs[4].Text = Mant(array5[i]);
}
```

```
}

// Конструктор:
// Этот конструктор вызывается для отчёта по обращению матриц,
// array1 - порядки матриц,
// array2 - время выполнения 1 способом,
// array3 - время выполнения 2 способом,
// array4 - погрешность 1 способом,
// array5 - погрешность 2 способом,
// array6 - число операций 1 способом,
// array7 - число операций 2 способом,
// array8 - теоретическое число операций.
public MyTable(int[] array1, double[] array2,
               double[] array3,
               double[] array4, double[] array5, double[]
array6,
               double[] array7, double[] array8) {
    double a = 1;
    do { eps = a; a /= 2; } while (a != 0);
    this.Padding = new Padding(0, 0, 0, 0);
    this.RowCount = array1.Length + 1;
    this.ColumnCount = 8;
    this.AutoSize = true;
    string[] titles = new string[] { "Порядок",
                                     "Время 1",
                                     "Время 2",
                                     "Погрешность 1",
                                     "Погрешность 2",
                                     "Реал. ЧО 1",
                                     "Реал. ЧО 2",
                                     "Теор. ЧО"};
    for (int i = 0; i < titles.Length; i++) {
        Label l = new Label();
        l.Text = titles[i];
        l.AutoSize = false;
        l.Width = 125;
        l.Height = 20;
```

```
l.Margin = new System.Windows.Forms.Padding(0, 0, 0, 0);
l.BackColor = System.Drawing.Color.DarkGray;
l.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
l.ForeColor = System.Drawing.Color.White;
this.Controls.Add(l);
}
for (int i = 0; i < array1.Length; i++) {
Label[] labs = new Label[8];
for (int j = 0; j < 8; j++) {
labs[j] = new Label();
labs[j].Margin = new
System.Windows.Forms.Padding(0, 0, 0, 0);
labs[j].AutoSize = false;
labs[j].TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
labs[j].BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle;
labs[j].BackColor = colors[i % 2];
labs[j].Width = 125;
labs[j].Height = 20;
this.Controls.Add(labs[j]);
}
labs[0].Text = array1[i].ToString();
labs[1].Text = Mant(array2[i]);
labs[2].Text = Mant(array3[i]);
labs[3].Text = Mant(array4[i]);
labs[4].Text = Mant(array5[i]);
labs[5].Text = Mant(array6[i]);
labs[6].Text = Mant(array7[i]);
labs[7].Text = Mant(array8[i]);
}
}

string Mant(double arg) {
int pow = 0;
if (arg >= 1 && arg < 1000) return
Math.Round(arg, 4).ToString();
```

```

if (arg >= 1000) {
    while (arg > 100) {
        arg /= 10;
        pow++;
    }
    return
    Math.Round(arg, 4).ToString() + "*" + pow.ToString() + ")";
}
if (arg < eps) arg = eps;
while (arg < 1) { pow++; arg *= 10; }
return
Math.Round(arg, 4).ToString() + "*" + (-10^pow.ToString());
}
}

```



Класс унаследован от уже существующего контейнера, имеющего табличную структуру.

Перейдём к классу, который будет рисовать графики. Создадим наследника от класса PictureBox. В конструктор будут передаваться заголовок графика и максимальный порядок матрицы, для которой формировались данные для отчёта. Весь исходный код класса GraphicPanel:

### **Листинг 6.67.**

```

// Класс отрисовки графиков
class GraphicPanel : PictureBox {
    int x_c, y_c, x_max, y_max;
    // В переменных будет записано значение начала координат и
    // максимальное значение в пикселях.
    Graphics g;
    // Переменная, через которую будет всё рисоваться на Bitmap.
    Bitmap btm;
    // Само изображение, которое будет выводить изображение на этот
    // экземпляр этого класса (Класс унаследован от PictureBox).
    int N_S;      // Начало отсчёта порядка матриц
    int N_E;      // Конец отсчёта порядка матриц
    int N_STEP;   // Шаг, с которым будет изменяться порядок матрицы
}

```

```
Label title = new Label();
// Метка с текстом, будет появляться при наведении пользователем
// указателя мыши на график. В тексте будет указано точно
// значение по оси Y для текущего положения курсора.

double strForText;
// В переменной будет сохранено значение по оси Y, которое
// приходится на 1 px

// Конструктор:
public GraphicPanel(String x_name, String y_name, int n_s,
                     int n_e, int n_step) {
    // Аргументы: x_name - метка оси X, y_name - метка оси Y,
    // n_s - начало отсчёта порядка матриц, n_e - конец отсчёта
    // порядка матриц, n_step - шаг изменения порядка матрицы
    titleBorderStyle =
        System.Windows.Forms.BorderStyle.FixedSingle;
    // Оформление и добавление метки с текстом на экземпляр этого
    // класса

    title.Visible = false;
    this.Controls.Add(title);
    this.MouseMove += GraphicPanel_MouseMove;
    // Присваиваем метод событию движения мыши по графику.

    N_S = n_s;
    // Сохраняем начало, конец отсчёта и шаг в глобальные переменные.
    N_E = n_e;
    N_STEP = n_step;
    this.Width = 600; // Устанавливаем размер графика
    this.Height = 600;
    x_c = 10;
    // Устанавливаем границы рабочей части графика.

    y_c = 400;
    x_max = 580;
    y_max = 10;
    this.BackColor = Color.White; // Делаем белый фон
                                // на графике.

    btm = new Bitmap(this.Width, this.Height);
    // Создаём Bitmap по размерам этого класса.
    g = Graphics.FromImage(btm); // Создаём экземпляр класса
```

```

    // Graphics.

// Рисуем оси, стрелки и название осей.
g.DrawLine(new Pen(Color.Black, 2), new Point(x_c, y_c),
new Point(x_c, y_max));
g.DrawLine(new Pen(Color.Black, 2), new Point(x_c, y_c),
new Point(x_max, y_c));
g.DrawLine(new Pen(Color.Black, 2), new Point(x_c, y_max),
new Point(x_c - 5, y_max + 5));
g.DrawLine(new Pen(Color.Black, 2), new Point(x_c, y_max),
new Point(x_c + 5, y_max + 5));
g.DrawLine(new Pen(Color.Black, 2), new Point(x_max, y_c),
new Point(x_max - 5, y_c + 5));
g.DrawLine(new Pen(Color.Black, 2), new Point(x_max, y_c),
new Point(x_max - 5, y_c - 5));
g.DrawString(x_name, new Font(x_name, 9),
new SolidBrush(Color.Blue), new PointF(x_c + 5, y_max - 5));
int x_ = x_max - y_name.Length * 6;
g.DrawString(y_name, new Font("Arial", 9),
new SolidBrush(Color.Blue), new PointF(x_, y_c - 20));
y_max += 10;
x_max -= 10;
this.Image = btm;

// Присваиваем изображению этого класса только что созданный
// Bitmap.
}

// Метод, вызываемый при движении курсора мыши по графику.
void GraphicPanel_MouseMove(object sender, MouseEventArgs e) {
    if (e.X > x_c && e.X < x_max && e.Y < y_c && e.Y > y_max) {
// Если курсор находится над рабочей частью графика, то делаем
// видимой подсказку,
        title.Visible = true;
        title.Top = e.Y + 5;
// присваиваем ей новое положение
        if (e.X + title.Width + 15 > this.Width) title.Left =
            e.X - title.Width - 5;
        else title.Left = e.X + 15;
        double val = (double)(y_c - e.Y) * strForText;
    }
}

```

```
// и записываем в неё новый текст.
if (val > 10 && val < 100000) title.Text =
    Math.Round(val, 4).ToString();
else title.Text = Mant(val);
}
else {
    title.Visible = false;
}
}

// Метод добавления одного графика
public void setData(double[] array1, String name1) {
// Аргументы: array1 - набор значений для отрисовки,
// name1 - имя графика для истории.
    double max = Max(array1);
// Ищем максимальное значение в массиве.
    strForText = max / 380;
// Вычисляем, какое значение приходится на один пиксель.
    LoadPanel(max);
// Создаём разметку.
    DrawGraphic(Color.Green, array1, max);
// Рисуем график
    DrawHistory(name1, Color.Green, x_c + 10, y_c + 20);
// Пишем пояснение к графику (историю).
}

// Метод добавления двух графиков.
public void setData(double[] array1, double[] array2,
                    String name1, String name2) {
// Аргументы: array1, array2 - наборы значений для двух графиков,
// name1, name2 - имена графиков.
    double max = Max(Max(array1), Max(array2));
// Ищем максимум из двух массивов.
    strForText = max / 380;
// Вычисляем, какое значение приходится на 1px.
    LoadPanel(max); //Делаем разметку.
    DrawGraphic(Color.Green, array1, max); // Рисуем графики.
    DrawGraphic(Color.Blue, array2, max);
    DrawHistory(name1, Color.Green, x_c + 10, y_c + 20);
```

```

// Рисуем историю.
    DrawHistory(name2, Color.Blue, x_c + 10, y_c + 45);
}
// Метод добавления трёх графиков.
public void setData(double[] array1, double[] array2,
                    double[] array3, String name1, String name2,
                                         String name3) {
// Аргументы: array1, array2, array3 - наборы значений для
// графиков; name1, name2, name3 - имена графиков.
    double max = Max(Max(Max(array1), Max(array2)), Max(array3));
// Поиск максимума из трёх графиков.
    strForText = max / 380;
// Вычисляем, какое значение приходится на 1px.
    LoadPanel(max);           // Делаем разметку.
    DrawGraphic(Color.Green, array1, max); // Рисуем графики.
    DrawGraphic(Color.Blue, array2, max);
    DrawGraphic(Color.Red, array3, max);
    DrawHistory(name1, Color.Green, x_c + 10, y_c + 20);
// Рисуем историю.
    DrawHistory(name2, Color.Blue, x_c + 10, y_c + 45);
    DrawHistory(name3, Color.Red, x_c + 10, y_c + 70);
}
// Метод рисует график
void DrawGraphic(Color color, double[] array, double max) {
// Аргументы: color - цвет графика, array - значения для
                                         // отрисовки,
// max - максимально возможное значение на всём графике.
    double stepY = 380 / max; // Вычисляем, сколько пикселей
// приходится на единицу в значениях графика по Y.
    int step_x = 560 / array.Length; // Вычисляем, сколько пикселей
// приходится на единицу в значениях графика по X.
    for (int i = 1; i < array.Length; i++) { // Рисуем график.
        int x1 = x_c + step_x * i;
// Вычисляем координаты текущей и предыдущей точек.
        int x2 = x_c + step_x * (i + 1);
        int y1 = y_c - (int)Math.Round(stepY * array[i - 1]);
        int y2 = y_c - (int)Math.Round(stepY * array[i]);
    }
}

```

```
        Draw(color, 2, x1, x2, y1, y2);      // Рисуем линию
    }
}

// Метод рисует строку и закрашенный цветной квадрат (пояснения
// к графику).
void DrawHistory(String str, Color color, int x, int y) {
// Аргументы: str - строка для отрисовки, color - цвет отрисовки
// квадрата, x, y - координаты.
    g.FillRectangle(new SolidBrush(color),
                    new Rectangle(new Point(x, y), new Size(20, 20)));
    g.DrawString("-" + str, new Font("Arial", 14),
                 new SolidBrush(Color.Black), new PointF(x + 24, y + 1));
}

// Метод выполняет разметку графика, а также подписывает значения
// на осях координат.
void LoadPanel(double max) {
// Аргументы: max - максимальное возможное значение на графике.
    double step_str = (max) / 10;
// Всего по оси Y будет 10 записей, вычисляем, сколько будет
// приходиться на одну.
    int l = 1;      // Заводим счётчик.
    int l_array = (N_E - N_S) / N_STEP;
// Вычисляем количество элементов в массиве.
    double stepX = 560 / (double)l_array;
// Вычисляем шаг по оси X в пикселях.
    int startStr = N_S;    // Сохраняем начало отсчёта.
    for (int i = 1; i <= l_array; i++) {
// Отрисовываем вертикальные полосы на графике и пишем к ним
// пояснения через одно (чтобы график был более читабельным).
        int x = x_c + (int)Math.Round((double)i * stepX);
        Draw(Color.LightGray, 1, x, x, y_c - 2, y_max);
        if (i % 2 == 1)
            DrawString(startStr.ToString(), x - 3, y_c + 3, 8);
        startStr += N_STEP;
    }
    for (int i = y_c - 38; i > y_max - 3; i -= 38) {
// Рисуем горизонтальные полосы и подписываем значения по оси Y.
```

```
Draw(Color.LightGray, 1, x_c + 1, x_max, i, i);
DrawString(Mant(step_str * l), x_c + 3, i - 1, 8);
l++;
}
}

// Метод вывода текста на график.
void DrawString(String str, int x, int y, float size) {
    g.DrawString(str, new Font("Arial", size),
        new SolidBrush(Color.Black), new PointF(x, y));
}

// Метод отрисовки прямой.
void Draw(Color color, int size, int x1, int x2, int y1, int y2)
{
    g.DrawLine(new Pen(color, size), new Point(x1, y1),
        new Point(x2, y2));
}

// Метод поиска максимума для двух значений.
double Max(double a, double b) { return a > b ? a : b; }

// Метод поиска максимума в массиве.
double Max(double[] arg) {
    double result = 0;
    for (int i = 0; i < arg.Length; i++) {
        if (arg[i] > result) result = arg[i];
    }
    return result;
}

// Метод перевода вещественного числа в строку.
string Mant(double arg) {
    int pow = 0;
    if (arg >= 1 && arg < 1000)
        return Math.Round(arg, 4).ToString();
    if (arg >= 1000) {
        while (arg > 100) {
            arg /= 10;
            pow++;
        }
        return Math.Round(arg, 4).ToString() + "*(" + 10^" +
    }
}
```

```
                pow.ToString() + ")";  
        }  
        while (arg < 1 && arg != 0) { pow++; arg *= 10; }  
        return Math.Round(arg, 4).ToString() + "*(-10^" +  
                           pow.ToString() + ")";  
    }  
}
```



Данный класс позволяет изображать до трёх графиков на одной координатной плоскости. Для выполнения задания такого количества графиков достаточно.

Теперь создадим класс, в который будем передавать данные о решении СЛАУ или об обращении матриц. Внутри него будут добавлены таблица и графики (это нужно сделать, чтобы дальше было меньше кода). Класс назовём OutputFlowLayoutPanel, и он будет унаследован от контейнера FlowLayoutPanel. Код этого класса приведён ниже.

### Листинг 6.68.

```
// Класс для отображения таблицы и графиков  
public class OutputFlowLayoutPanel : FlowLayoutPanel {  
    // Конструктор для передачи данных о решении СЛАУ  
    public OutputFlowLayoutPanel(int[] array1, double[] array2,  
                                double[] array3, double[] array4, double[] array5,  
                                double[] array6, double[] array7, double[] array8) {  
        this.AutoSize = true;  
        this.Controls.Add(new MyTable(array1, array2, array3, array4,  
                                      array5, array6, array7, array8));  
        if (array1.Length != 1) {  
            GraphicPanel graph1 = new GraphicPanel("Число операций",  
                                         "Порядок матрицы", array1[0], array1[array1.Length - 1],  
                                         array1[1] - array1[0]);  
            GraphicPanel graph2 = new GraphicPanel("Время выполнения",  
                                         "Порядок матрицы", array1[0], array1[array1.Length - 1],  
                                         array1[1] - array1[0]);  
            GraphicPanel graph3 = new GraphicPanel("Погрешность решения",  
                                         "Порядок матрицы", array1[0], array1[array1.Length - 1],  
                                         array1[1] - array1[0]);  
        }  
    }  
}
```

```

graph1.setData(array6, array7, array8, "Реал. Ч0 сп.1",
    "Реал. Ч0 сп.2", "Теор. Ч0");
graph2.setData(array2, array3, "Время сп.1", "Время сп. 2");
graph3.setData(array4, array5, "Погрешность сп.1",
    "Погрешность сп.2");
this.Controls.Add(graph1);
this.Controls.Add(graph2);
this.Controls.Add(graph3);
}

}

// Конструктор для передачи данных об обращении матриц.
public OutputFlowLayoutPanel(int[] array1, double[] array2,
    double[] array3, double[] array4, double[] array5) {
    this.AutoSize = true;
    this.Controls.Add(new MyTable(array1, array2, array3, array4,
        array5));
    if (array1.Length != 1) {
        GraphicPanel graph1 = new GraphicPanel("Число операций",
            "Порядок матрицы", array1[0], array1[array1.Length - 1],
            array1[1] - array1[0]);
        GraphicPanel graph2 = new GraphicPanel("Время выполнения",
            "Порядок матрицы", array1[0], array1[array1.Length - 1],
            array1[1] - array1[0]);
        GraphicPanel graph3 = new GraphicPanel("Погрешность решения",
            "Порядок матрицы", array1[0], array1[array1.Length - 1],
            array1[1] - array1[0]);
        graph1.setData(array4, array5, "Теоретическое Ч0",
            "Фактическое Ч0");
        graph2.setData(array2, "Время выполнения");
        graph3.setData(array3, "Погрешность решения");
        this.Controls.Add(graph1);
        this.Controls.Add(graph2);
        this.Controls.Add(graph3);
    }
}
}

```

Всё готово для того, чтобы собрать весь проект, а именно: подключить созданные ранее dll библиотеки и формировать отчёты. Для подключения библиотек необходимо в главном меню выбрать пункт «Проект», далее – пункт «Добавить ссылку», нажать «Обзор» и указать путь к нашим библиотекам (рис. 6.23), чтобы их выбирать (рис. 6.24).

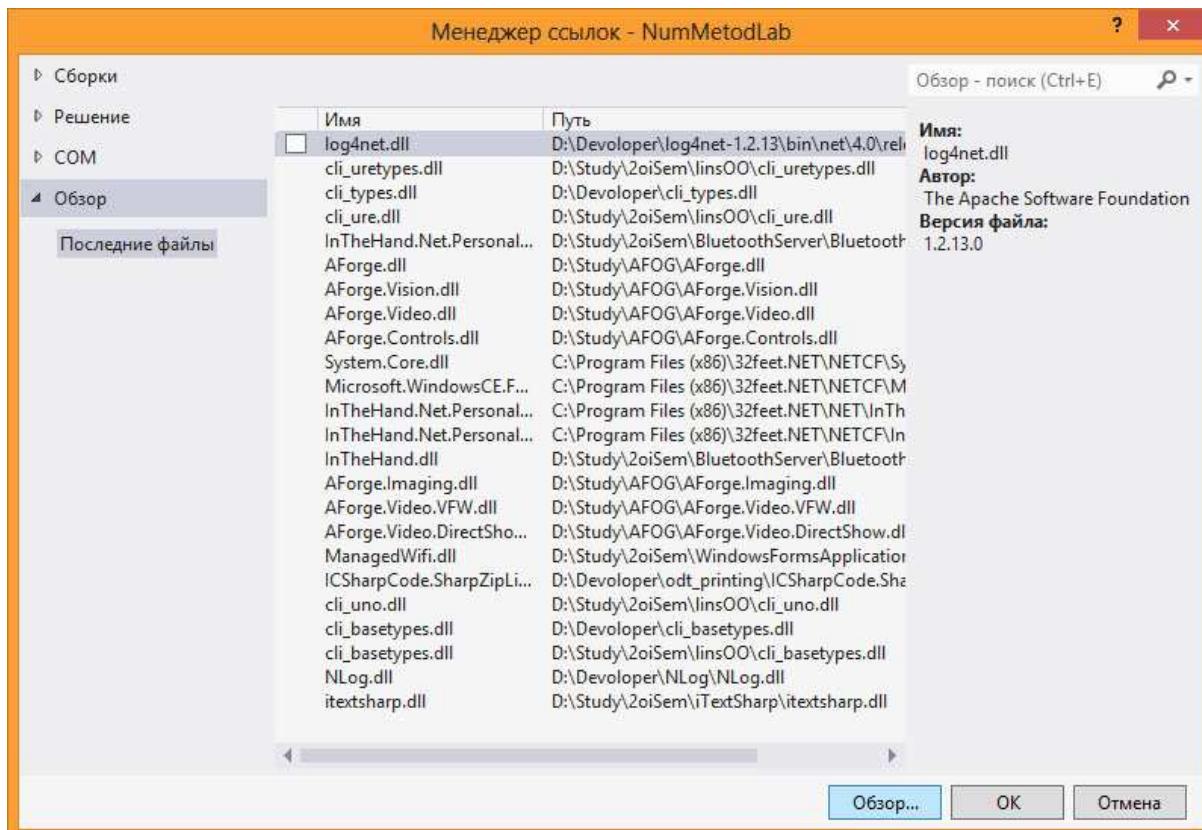


Рис. 6.23. Подключение библиотек

После того как библиотеки подключены к проекту, возвращаемся к классу OptionMatrixPanel, в котором будем редактировать метод, срабатывающий при нажатии пользователем кнопки. Предлагаем в зависимости от выбранного пункта в ListBox при нажатии на кнопку вызывать разные методы:

### Листинг 6.69.

```
void btn_go_Click(object sender, EventArgs e) {
    switch (type) {
        case 1:
            GenerateSLAU();
            break;
        case 2:
            break;
    }
}
```

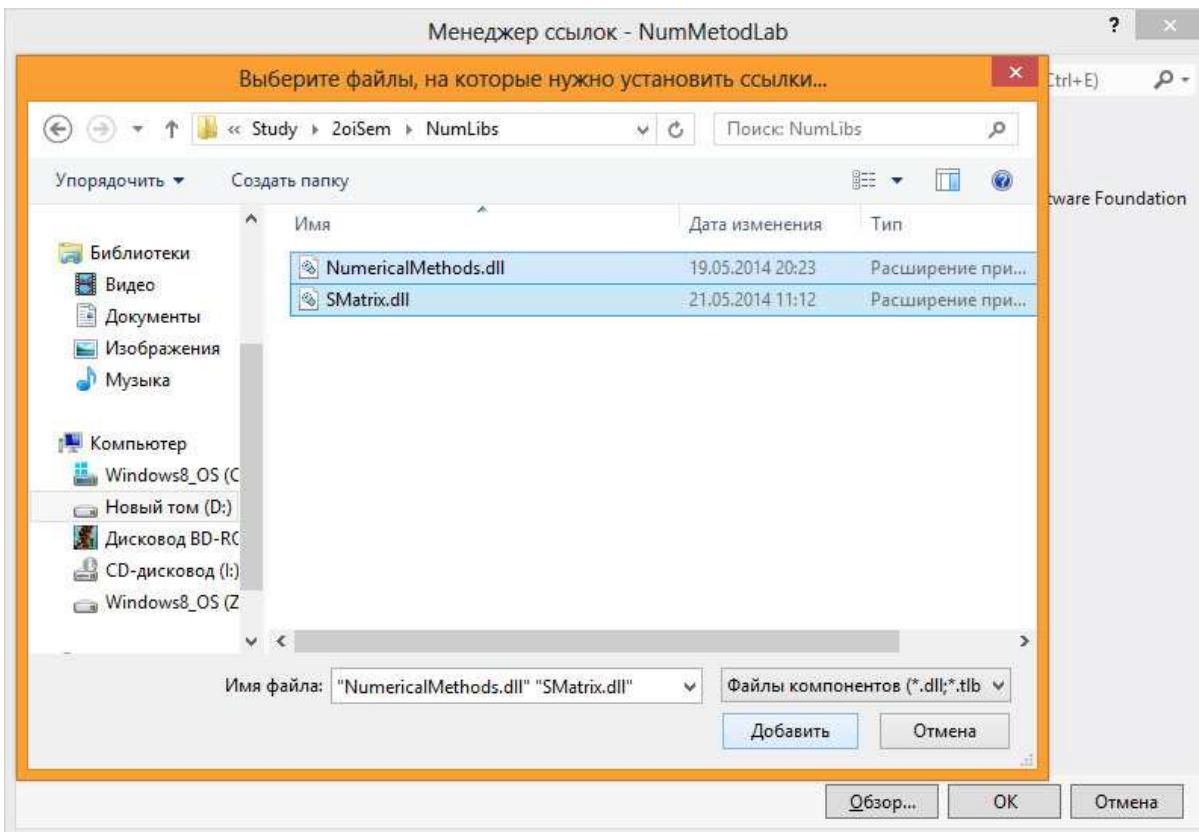


Рис. 6.24. Выбор библиотек

```

GenerateInv();
break;
case 3:
CalcDet();
break;
}
}

```

Если выбран пункт «Решение СЛАУ», то вызывается метод `GenerateSLAU`. Для других пунктов – аналогично. В самих методах (за исключением вычисления определителя, там ввод всегда выполняется вручную) в первую очередь будет проверяться способ ввода матрицы (вручную, случайные или специальные матрицы). Если ввод выбран ручной, то вызываем соответствующую форму, которую создали ранее. Если ввод выбран другой, то определяем его вместе с необходимыми для генерации данными, и решаем поставленную задачу с помощью классов, которые были созданы ранее и подключены как dll библиотеки.

Приведём пример кода метода `GenerateSLAU` с обработкой ручного ввода:

**Листинг 6.70.**

```
// Метод ввода матриц для решения СЛАУ.
void GenerateSLAU() {
    int[] arrayN = null;      // Массив порядков матриц.
    double[] array1 = null;   // Массив времени выполнения.
    double[] array2 = null;   // Массив погрешности.
    double[] array3 = null;   // Массив теоретического числа операций.
    double[] array4 = null;   // Массив фактического числа операций.
    Label tit = null;         // Метка для вывода ответа.
    int start_n = Convert.ToInt32(int_ot_txt.Text);
    // Переводим значения текстовых полей в числовой формат.
    int end_n = Convert.ToInt32(int_do_txt.Text);  //
    int step_n = Convert.ToInt32(step_txt.Text);
    double ar = Convert.ToDouble(arg.Text.Replace('.', ','));
    switch (comb[0].SelectedIndex) {
        case 0:{  

            HandInputSLAU handSlau = new HandInputSLAU();
            // Создаём форму с вводом матрицы.
            if (handSlau.ShowDialog() == DialogResult.OK) {
                // Открываем её в режиме диалога.
                arrayN = new int[1];    // Так как матрица вводится одна,
                // то и размеры массивов будут равны единице.
                array1 = new double[1];
                array2 = new double[1];
                array3 = new double[1];
                array4 = new double[1];
                NumMeth num = new NumMeth(EPS);
                // Создаём экземпляр класса NumMeth.
                num.setA(handSlau.A, handSlau.N);
                // Вызываем метод LU-разложения.
                if (num.flagError) {    // Проверям флаг ошибки.
                    MessageBox.Show("Произошло деление на ноль! \r\nРешение данной
                        системы через LU-разложение невозможно!");
                    return;
                }
                // Читаем остальные введённые данные.
                double[] X_toch = handSlau.X;
```

```

double[] B_toch = handSlau.B;
double[] X = num.getX(B_toch); // Решаем СЛАУ.
array1[0] = num.TIME;           // Фиксируем данные для отчёта.
array3[0] = num.OPER_T;
array4[0] = num.OPER_F;
if (X_toch != null) {
// Если X_toch был введён, то вычисляем погрешность, сравнивая эти
// векторы.
    array2[0] = num.VectorPogr(X_toch, X);
}
else {
// Если X_toch введён не был, то погрешность находим через
// сравнение правых частей СЛАУ.
    double[] B = num.getB(handSlau.A, X, handSlau.N);
    array2[0] = num.VectorPogr(B, B_toch);
}
arrayN[0] = handSlau.N; // Запоминаем порядок матрицы.
tit = new Label();      // Записываем решение.
tit.Text = "X = [ ";
tit.AutoSize = false;
tit.Width = 800;
for (int i = 0; i < handSlau.N; i++) {
    tit.Text += Mant(X[i]);
    if (i != handSlau.N - 1) tit.Text += " ; ";
}
tit.Text += " ]";
}
else {
    return;
}
}
break;
case 1: {} break;
case 2:{} break;}
// Создаём панель для вывода отчёта.
OutputFlowLayoutPanel output_panel =
new OutputFlowLayoutPanel(arrayN, array1, array2, array3, array4);

```

```
// Создаём новую форму.
FormOutput output_form = new FormOutput();
// Добавляем на форму панель.
output_form.flowLayoutPanel1.Controls.Add(output_panel);
// Если был записан ответ, то его также добавляем на форму.
if (tit != null) {
    Panel otvet = new Panel();
    otvet.Location = new System.Drawing.Point(10, 100);
    otvet.Controls.Add(tit);
    otvet.Width = 800;
    output_form.flowLayoutPanel1.Controls.Add(otvet);
}
// Открываем форму в режиме диалога
output_form.ShowDialog();
}
```



Если выбор пользователя остановится на единице, то были выбраны случайные матрицы. Если на двойке, – то были выбраны специальные матрицы. Для обоих этих случаев необходимо изображать дополнительно три графика. Следующий код написан для случайных матриц:

### Листинг 6.71.

```
case 1: {
    arrayN = new int[(end_n - start_n) / step_n+1];
    // Вычисляем и присваиваем длину массива.
    array1 = new double[arrayN.Length];
    array2 = new double[arrayN.Length];
    array3 = new double[arrayN.Length];
    array4 = new double[arrayN.Length];
    int l = 0;           // Заводим счётчик.
    for (int i = start_n; i <= end_n; i += step_n) {
        arrayN[l] = i;      // Запоминаем размер матрицы.
        SpecMatrix matrix; // Создаём экземпляр SpecMatrix.
        NumMeth lu=null;   // Создаём экземпляр NumMeth.
        do {
            lu = new NumMeth(EPS);
        // Так как при случайных матрицах может произойти деление на ноль,
```

```

// то выполняем этот цикл до тех пор, пока не будет найдена
// подходящая матрица.
matrix = new SpecMatrix(i, 11);
lu.setA(matrix.Matrix, i);
} while(lu.flagError);
double[] X_toch = new double[i];
// Создаём точное решение системы.
for (int j = 0; j < i; j++) X_toch[j] = j + 1;
double[] B = lu.getB(matrix.Matrix, X_toch, i);
// На основе точного решения вычисляем B.
double[] X = lu.getX(B); // Вычисляем вектор X.
array1[1] = lu.TIME; // Фиксируем данные для отчёта.
array2[1] = lu.VectorPogr(X_toch, X);
array3[1] = lu.OPER_T;
array4[1] = lu.OPER_F;
l++; // Увеличиваем счётчик.
}
}
break;

```



Аналогично формируются отчёты для всех остальных типов специальных матриц. Посмотреть весь код можно, скачав готовый проект по ссылке в конце отчёта.

Осталось вычислить определитель. Для этого перейдём к методу CalcDet, который вызывается при нажатии на кнопку, если из списка выбран третий пункт («Вычисление определителя»). Вычисление определителя будет производиться только для матриц, введённых вручную, поэтому сначала необходимо считать матрицу, введённую в текстовое поле. После этого передаём её в метод setA класса NumMeth. Далее метод getDet всё того же класса вернёт значение определителя. Ниже приведён код, выполняющий все эти операции.

### Листинг 6.72.

```

void CalcDet() {
    string[] rows = matrix_input.Text.Split(new char[] { '\r' });
    double[,] A = new double[rows.Length, rows.Length];
    try {

```

```
for (int i = 0; i < rows.Length; i++) {
    rows[i] = rows[i].Replace("\n", "");
    string[] cols = rows[i].Split(new char[] { ' ' });
    if (cols.Length != rows.Length) {
        MessageBox.Show("Данные введены некорректно!");
        return;
    }
    for (int j = 0; j < cols.Length; j++) {
        A[i, j] = Convert.ToDouble(cols[j].Replace(".", ","));
    }
}
NumMeth meth = new NumMeth();
meth.setA(A);
double det = meth.getDet();
MessageBox.Show("Определитель равен: "+
    Math.Round(det, 5).ToString());
}
catch (Exception e) {
    MessageBox.Show("Данные введены некорректно!");
}
}
```



## 6.5 Завершающее тестирование

Начнём тестирование с проверки решения СЛАУ в режиме ручного ввода матриц (рис. 6.25). Этот режим – отладочный. Он нужен для того, чтобы убедиться, что алгоритмы решения реализованы правильно.

В качестве тестируемых матриц рекомендуем брать те матрицы, для которых вы «вручную» нашли точное решение основных трёх задач: решение СЛАУ, обращение матрицы и вычисление определителя. Наборы задач для тестирования приведены в конце разд. 3.

Для задачи, введённой согласно рис. 6.25, убеждаемся в правильности разработанной программы решения СЛАУ, поскольку мы заранее «вручную» нашли точное решение этой задачи (рис. 6.26).

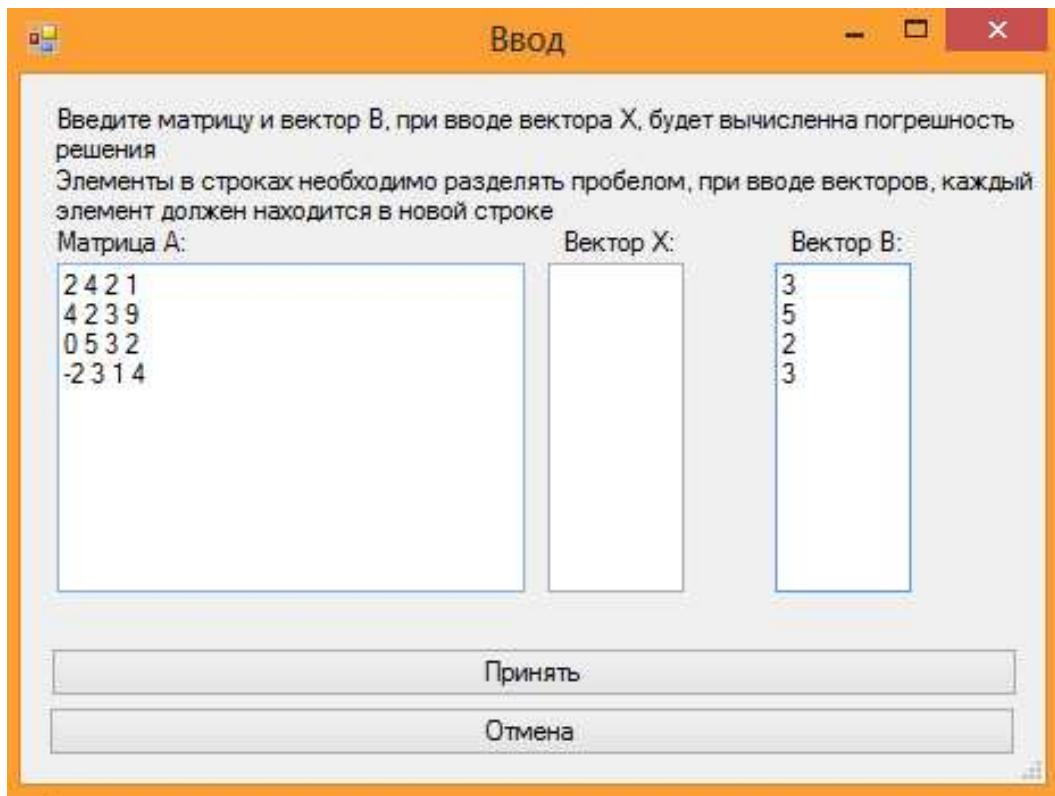


Рис. 6.25. Ввод матриц для решения СЛАУ

Выход результата				
Порядок	Время	Точность	Теор. ЧО	Факт. ЧО
4	4,9407*(-10^324)	8,8818*(-10^16)	80	42
$X = [5,5882*(-10^1); 1,1176; 4,9407*(-10^324); 5,8824*(-10^1)]$				

Рис. 6.26. Решение СЛАУ и отчёт в таблице для задачи из рис. 6.25

Тестирование разработанных программ можно проводить и по-иному: сравнивая решение, получаемое в нашей программе, с решением, которое мы получаем из другой, заведомо верной программы, принимаемой за эталон. В качестве программы-эталона рекомендуем пользоваться пакетом MATLAB.

Удостоверившись в правильной работе программы решения СЛАУ, перейдём к вычислительным экспериментам со специальными (плохо обусловленными) матрицами в режимах «Решение СЛАУ» и «Обратные матрицы» (два способа обращения). В первом режиме используем матрицу Гильберта (первый тип специальных матриц). Во втором режиме используем специальную матрицу девятого типа. Это даст возможность проверить:

- работу программ вывода таблицы результатов, рис. 6.27; по этим таблицам должны строиться графики,
- вывод графика количества операций в зависимости от порядка матрицы от 5 до 100 с шагом 5, рис. 6.28,
- вывод графика времени выполнения в зависимости от порядка матрицы от 5 до 100 с шагом 5, рис. 6.29,
- вывод графика погрешности в зависимости от порядка матрицы от 5 до 100 с шагом 5, рис. 6.30.

Завершим отладочное тестирование проверкой программы вычисления определителя применительно к матрице, введённой с клавиатуры, рис. 6.31.

## 6.6 Заключение по разделу 6

В этом разделе мы продемонстрировали реальный пример лабораторного проекта № 1 «Стандартные алгоритмы  $LU$ -разложения», выполненного в весенном семестре 2014 года на кафедре «Информационные системы» Ульяновского государственного технического университета.

Из этого примера, содержащего 72 листинга, видно, что объём работы студента над таким проектом достаточно велик. Мы оцениваем трудоёмкость такого проекта средней продолжительностью работы над ним. Естественно, срок зависит от ряда факторов, но прежде всего, он определяется степенью организованности студента. Для студента дневного отделения трудоёмкость проекта № 1 оценивается величиной до двух месяцев регулярной работы.

Приведённое описание проекта является рекомендательным. В нём выделены следующие пять подразделов:

1. Постановка задачи (подразд. 6.1, с. 221).
2. Класс с реализацией алгоритмов (подразд. 6.2, с. 223).
3. Генерация матриц (подразд. 6.3, с. 248).
4. Создание пользовательского интерфейса и подключение ранее созданных библиотек (подразд. 6.4, с. 266).
5. Завершающее тестирование (подразд. 6.5, с. 306).

Порядок	Время	Точность	Теор. ЧО	Факт. ЧО
5	4,9407*(-10^324)	3.5583*(-10^11)	150	72
10	4,9407*(-10^324)	3.3747*(-10^3)	11*(10^2)	463
15	4,9407*(-10^324)	181,4944	36*(10^2)	14,28*(10^2)
20	4,9407*(-10^324)	519,4808	84*(10^2)	32,13*(10^2)
25	1,0006	16,8606*(10^2)	16,25*(10^3)	60,73*(10^2)
30	4,9407*(-10^324)	16,5614*(10^2)	27,9*(10^3)	10,256*(10^3)
35	4,9407*(-10^324)	25,2624*(10^2)	44,1*(10^3)	16,008*(10^3)
40	1,0003	21,7772*(10^2)	65,6*(10^3)	23,576*(10^3)
45	1,0007	47,6519*(10^2)	93,15*(10^3)	33,238*(10^3)
50	1,0006	27,6194*(10^2)	12,75*(10^4)	45,205*(10^3)
55	1,0007	25,0437*(10^2)	16,94*(10^4)	59,76*(10^3)
60	2,0013	69,3273*(10^2)	21,96*(10^4)	77,139*(10^3)
65	3,002	34,9007*(10^3)	27,885*(10^4)	97,561*(10^3)
70	3,002	71,1167*(10^2)	34,79*(10^4)	12,1317*(10^4)
75	3,002	91,3389*(10^2)	42,75*(10^4)	14,8666*(10^4)
80	4,0027	19,8625*(10^3)	51,84*(10^4)	17,9839*(10^4)
85	4,0027	35,5073*(10^3)	62,135*(10^4)	21,5065*(10^4)
90	5,0033	12,4916*(10^4)	73,71*(10^4)	25,4598*(10^4)
95	5,0033	45,3429*(10^3)	86,64*(10^4)	29,8753*(10^4)
100	7,005	54,3713*(10^3)	10,1*(10^5)	34,7727*(10^4)

(a)

Порядок	Прим. 1	Прим. 2	Первичность 1	Первичность 2	Рез. ЧО 1	Рез. ЧО 2	Тип. ЧО
5	4,9407*(-10^324)	4,9407*(-10^324)	3,638*(-10^12)	7,275*(-10^12)	192	187	375
10	4,9407*(-10^324)	4,9407*(-10^324)	8,3923*(-10^5)	7,6294*(-10^5)	14,28*(-10^2)	12,43*(-10^2)	30*(10^2)
15	1,0007	4,9407*(-10^324)	1,25	82	47,13*(-10^2)	39,23*(-10^2)	10,125*(-10^3)
20	4,9407*(-10^324)	4,9407*(-10^324)	3,625	49,5	11,043*(-10^3)	89,73*(-10^2)	24*(-10^3)
25	1,0002	4,9407*(-10^324)	7	66	21,423*(-10^3)	17,148*(-10^3)	46,875*(-10^3)
30	4,9407*(-10^324)	1,0003	8,375	578,6875	36,851*(-10^3)	29,196*(-10^3)	81*(-10^3)
35	1,0003	1,0006	4,25	13,4081*(-10^2)	58,323*(-10^3)	45,863*(-10^3)	12,8625*(-10^4)
40	1,0007	1,0006	4,9375	570,25	86,836*(-10^3)	67,896*(-10^3)	19,2*(-10^4)
45	2,0013	2,0013	14,4375	270,625	12,3418*(-10^4)	96,073*(-10^3)	27,3375*(-10^4)
50	2,0021	2,0013	19,875	475,75	16,903*(-10^4)	13,1105*(-10^4)	37,5*(-10^4)
55	3,002	3,002	26	203	22,4705*(-10^4)	17,3775*(-10^4)	49,9125*(-10^4)
60	4,0022	4,0027	21,5	173,5	29,1429*(-10^4)	22,4819*(-10^4)	64,8*(-10^4)
65	5,0033	5,0033	20,25	435,0625	37,0171*(-10^4)	28,49561*(-10^4)	82,3875*(-10^4)
70	6,004	7,0046	8,875	292,5	46,1972*(-10^4)	35,4977*(-10^4)	10,29*(-10^5)
75	7,0047	8,0053	24	601,75	56,7841*(-10^4)	43,5641*(-10^4)	12,6562*(-10^5)
80	9,006	10,0066	27	301,125	68,8759*(-10^4)	52,7679*(-10^4)	15,36*(-10^5)
85	11,0077	12,0079	11,25	11,5079*(-10^2)	82,5705*(-10^4)	63,182*(-10^4)	18,4238*(-10^5)
90	12,008	14,0097	47	21,1875*(-10^2)	97,9683*(-10^4)	74,8818*(-10^4)	21,87*(-10^5)
95	14,0092	16,0107	28	14,9425*(-10^2)	11,5176*(-10^5)	87,9488*(-10^4)	25,7212*(-10^5)
100	18,0119	18,0116	15,75	582,875	13,4288*(-10^5)	10,2452*(-10^5)	30*(-10^5)

(6)

Рис. 6.27. Вывод таблиц результатов эксперимента: (а) в режиме «решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «обратная матрица» при использовании плохо обусловленных матриц девятого типа

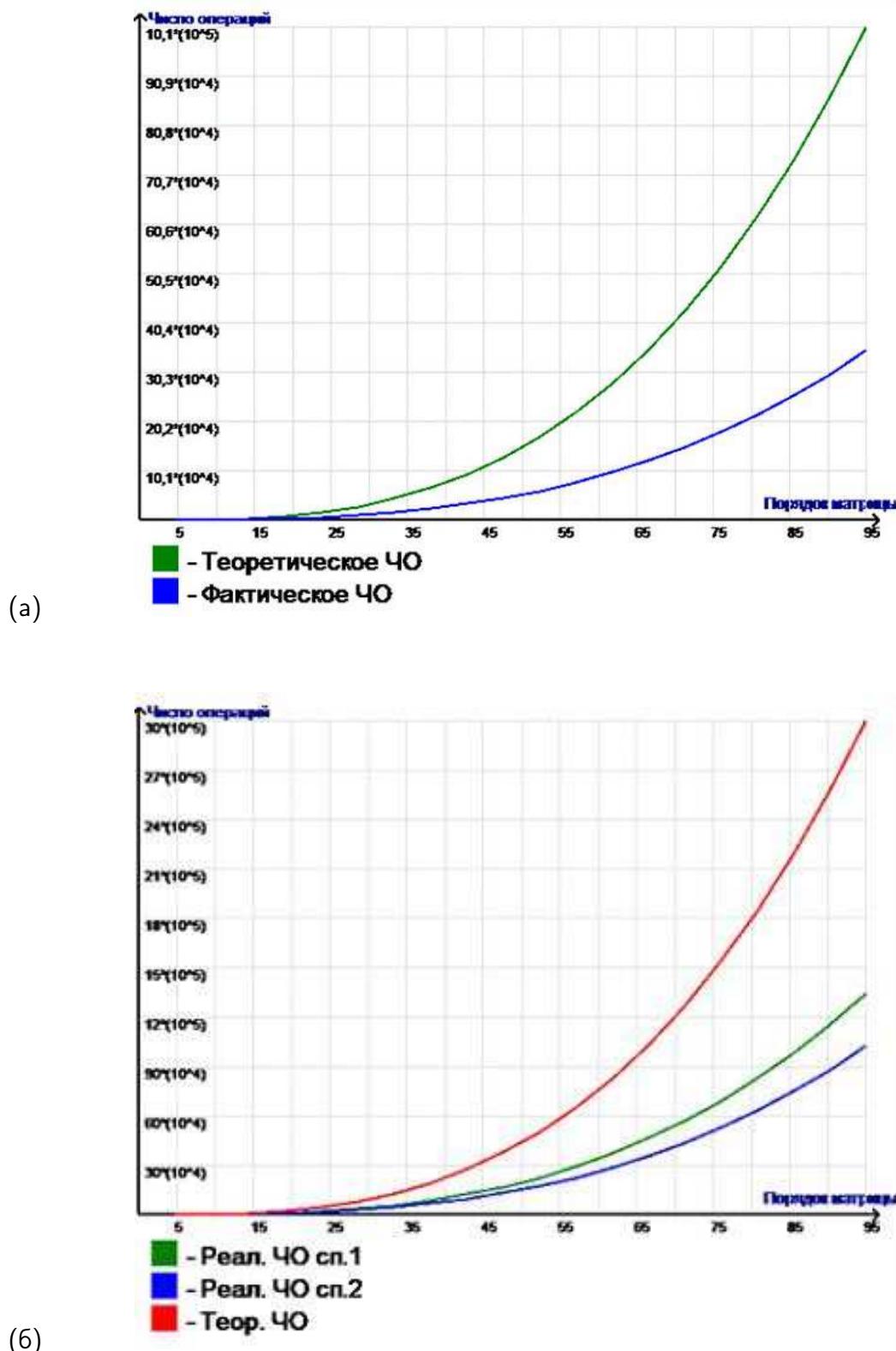
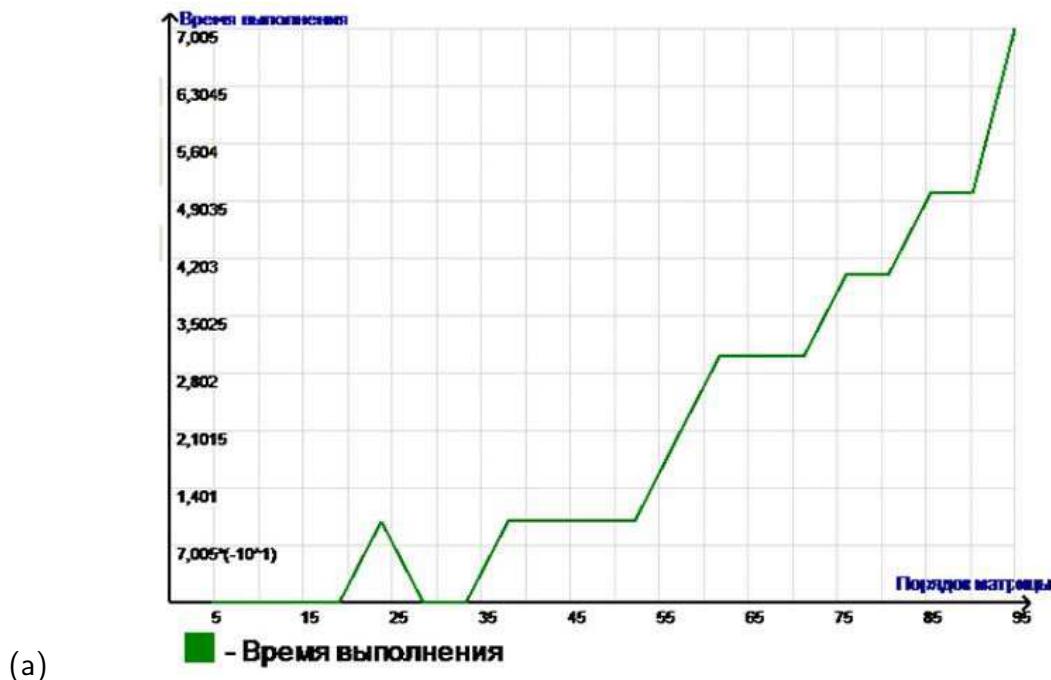
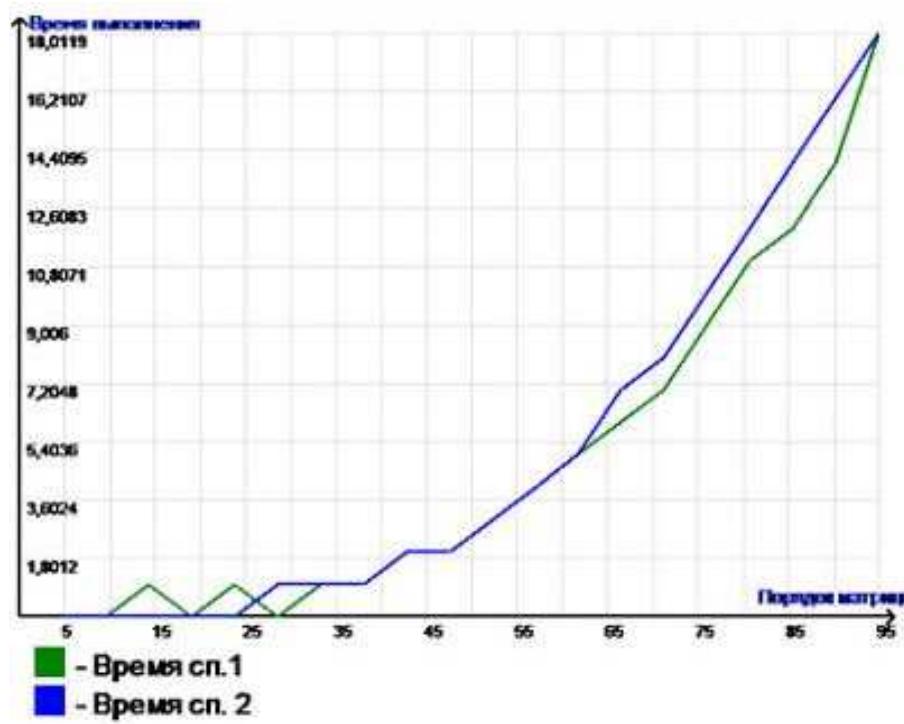


Рис. 6.28. Вывод графика зависимости числа операций от порядка матрицы: (а) в режиме «Решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «Обратная матрица» при использовании плохо обусловленных матриц девятого типа



(а)



(б)

Рис. 6.29. Вывод графика зависимости времени выполнения от порядка матрицы: (а) в режиме «Решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «Обратная матрица» при использовании плохо обусловленных матриц девятого типа

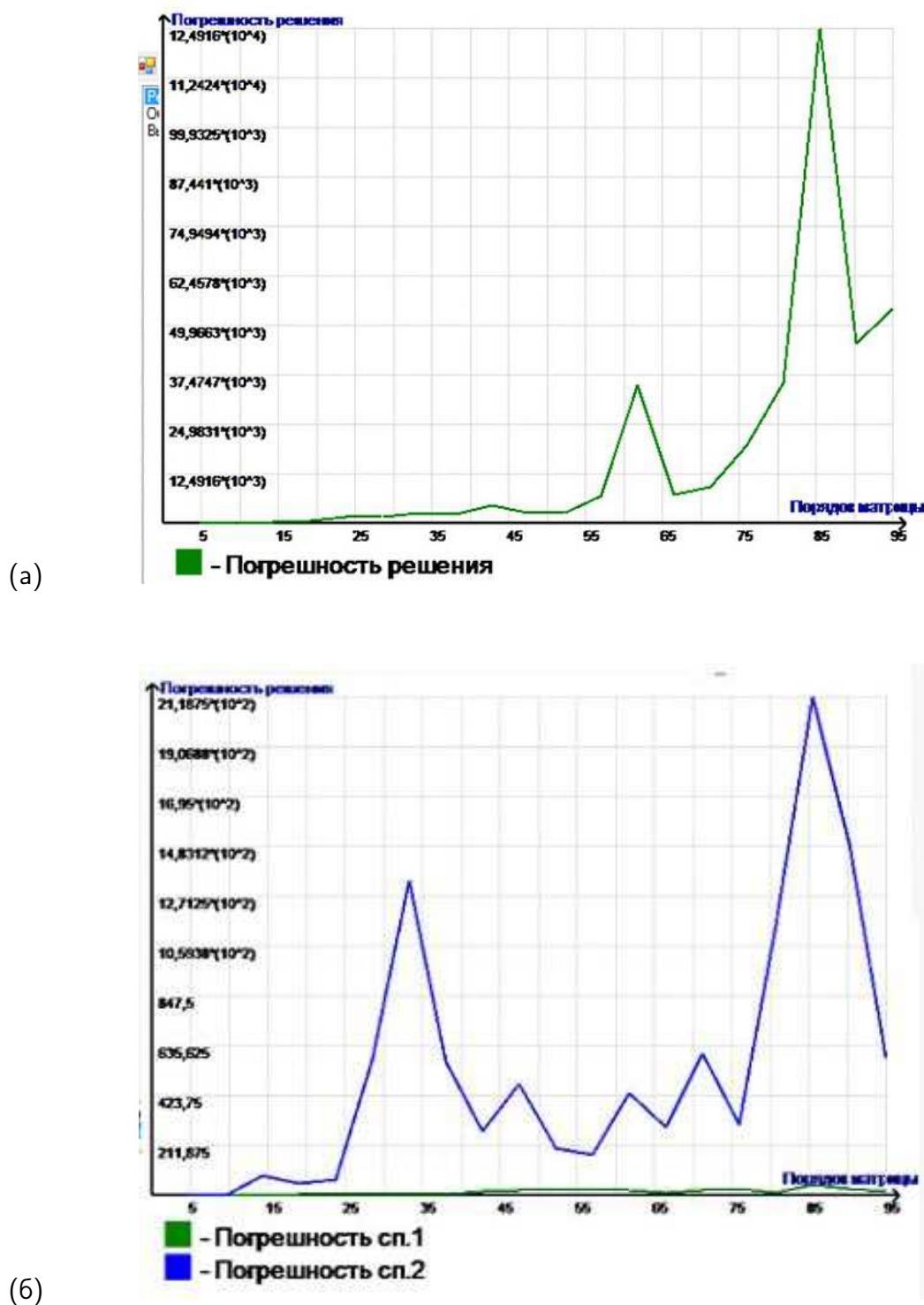


Рис. 6.30. Вывод графика зависимости погрешности от порядка матрицы: (а) в режиме «Решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «Обратная матрица» при использовании плохо обусловленных матриц девятого типа

Вычисление определителя

Введите матрицу, для вычисления определителя(элементы в строке разделяйте пробелом, каждая строка матрицы вводится с новой строчки)

2 3 -2 1 9 2 3 4 1 2 3 5 -10 1 1 2 -4
--

(a)

Вычисление определителя

Введите матрицу, для вычисления определителя(элементы в строке разделяйте пробелом, каждая строка матрицы вводится с новой строчки)

2 3 -2 1 9 2 3 4 1 2 3 5 -10 1 1 2 -4
--

Определитель равен: 1984

(б)

Рис. 6.31. Эксперимент с вычислением определителя заданной матрицы: (а) Ввод матрицы и (б) Результат вычисления определителя

Приведённое в этом разделе описание включает множество полезных примеров, которые помогут студентам выполнить свои индивидуальные проекты.

Рекомендуемые здесь приёмы программной реализации проекта № 1 не являются ни обязательными, ни единственными верными. Здесь представлен лишь один из возможных вариантов решения задачи проектирования.

**Ссылка на выполненный проект:**

[https://drive.google.com/folderview?  
id=0B56qWLWvTkWUVk5UZk03czVNNOU&usp=sharing](https://drive.google.com/folderview?id=0B56qWLWvTkWUVk5UZk03czVNNOU&usp=sharing) (перенос адреса)  
скопируйте  
весь адрес в строку браузера

# III

# ПОВЫШЕННЫЙ КУРС



# 7

## Проект № 4 «Векторно-ориентированные версии $LU$ -разложения»

### 7.1 Гауссово исключение и $ijk$ -алгоритмы

Рассмотрим систему линейных уравнений

$$Ax = b \quad (7.1)$$

с невырожденной матрицей  $A$  размера  $(n \times n)$ . Мы считаем  $A$  заполненной матрицей. Разреженные матрицы рассматриваются ниже в разд. 10.

Как изложено в подразд. 3.1, наиболее известной формой гауссова исключения является та, в которой система (7.1) приводится к верхнетреугольному виду путём вычитания одних уравнений, умноженных на подходящие числа, из других уравнений; полученная треугольная система решается с помощью обратной подстановки. Математически всё это эквивалентно тому, что вначале строится разложение матрицы  $A$ , например вида  $A = \bar{L}U$ , где  $\bar{L}$  является нижнетреугольной матрицей с единицами на главной диагонали, а  $U$  – верхнетреугольная матрица с ненулевыми элементами на диагонали. Затем решаются треугольные системы

$$\bar{L}y = b, \quad Ux = y. \quad (7.2)$$

Процесс их решения называется, соответственно, *прямой* и *обратной* подстановками.

Мы сосредоточимся вначале на  $\bar{L}U$ -разложении, поглощающем большую часть времени всего процесса, а затем вернёмся к решению треугольных систем. Псевдокод разложения приведён на рис. 7.1.

```

Для  $k = 1$  до  $n - 1$ 
    Для  $i = k + 1$  до  $n$ 
         $l_{ik} = a_{ik}/a_{kk}$ 
        Для  $j = k + 1$  до  $n$ 
             $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 

```

Рис. 7.1. Строчно ориентированная схема  $\bar{L}U$ -разложения

```

Для  $k = 1$  до  $n - 1$ 
    Для  $s = k + 1$  до  $n$ 
         $l_{sk} = a_{sk}/a_{kk}$ 
        Для  $j = k + 1$  до  $n$ 
            Для  $i = k + 1$  до  $n$ 
                 $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 

```

Рис. 7.2. Столбцово ориентированная схема  $\bar{L}U$ -разложения

В цикле  $j$  на рис. 7.1 кратные  $k$ -й строки текущей матрицы  $A$  вычтутся из расположенных ниже строк. Эти операции представляют собой *триады*, в которых векторами являются строки матрицы  $A$  [9].

**Определение 7.1.** *Триадой* называют операцию вида  $\mathbf{a} + \alpha\mathbf{b}$ , где  $\mathbf{a}$  и  $\mathbf{b}$  – векторы, а  $\alpha$  – скаляр<sup>1</sup>.

Определение триады появилось в связи с использованием векторных компьютеров<sup>2</sup>, требующих, чтобы векторы располагались в последовательно адресуемых ячейках памяти. Для алгоритма на рис. 7.1 удобно предположить, что  $A$  хранится по строкам. Соответственно этому, схема на рис. 7.1 названа строчно ориентированной. В ней посредством триад осуществляется *модификация* (т. е., *обновление*) строк матрицы  $A$ ; на это приходится основная часть работы в  $L$  $U$ -разложении.

Реальные задачи, как правило, требуют выбора главного элемента, и это ещё сильнее уменьшает скорость. При использовании одной из стратегий частичного выбора мы должны на первом шаге просмотреть первый столбец в поисках максимального по модулю элемента. Эта стратегия, соответственно, называется *выбором главного элемента по столбцу*, и она приводит к перестановке строк<sup>3</sup>. Как только положение максимального элемента определено, соответствующую строку можно переставить с первой (точнее, текущей ведущей) строкой или изменить порядок индексации строк. Второй вариант называют *неявной перестановкой* строк. Как именно реализуется стратегия выбора главного элемента, зависит от вашего варианта задания. Однако во всех вариантах лабораторных работ физическая, т. е., явная перестановка строк (или столбцов) запрещена и должна быть заменена изменением порядка нумерации строк (или столбцов), т. е., неявной перестановкой. Это требование соответствует реальным пакетам

<sup>1</sup> В зарубежной литературе триаду называют также *saxpy*, что обозначает операцию  $y := ax + y$  и заменяет фразу: Single precision (с обычной точностью)  $ax$  Plus  $y$ .

<sup>2</sup> По поводу триады и векторных компьютеров см. подразд. 7.2 и 7.3.

<sup>3</sup> Подробнее о стратегиях выбора главного элемента см. подразд. 3.2.

вычислительной линейной алгебры. т. е., так в реальности всегда и делают. В схеме на рис. 7.1 возможны все три стратегии выбора главного элемента (см. подразд. 3.2).

Правая часть  $b$  системы (7.1) также может обрабатываться в ходе приведения системы к треугольному виду, благодаря чему осуществляется этап прямой подстановки в равенствах (7.2). Такая обработка правой части  $b$ , выполняемая одновременно с приведением матрицы  $A$  к треугольному виду, также запрещена во всех вариантах лабораторных работ (проектов). Это требование тоже отвечает реальности, так как позволяет экономить значительное время в условиях, когда требуется решать одну и ту же систему (7.1) с различными правыми частями. Ситуация такого рода – обращение матрицы с помощью решения матричного уравнения  $AX = I$ , где  $I$  – единичная матрица, т. е., нахождение  $X = A^{-1}$ . В этом случае правые части  $b$  вводятся в уравнения (7.2) последовательно. При вычислении  $i$ -го столбца матрицы  $X = A^{-1}$  каждая правая часть равна очередному ( $i$ -му) столбцу единичной матрицы. Для каждого  $b$  в (7.2) сначала решают первую систему, т. е., вычисляют  $y$ , а затем – вторую систему, т. е., вычисляют  $x$ . Существенно, что для хранения ни  $y$ , ни  $x$  затрат памяти не требуется: их можно хранить там же, куда был введён вектор  $b$ .

Если  $A$  хранится по столбцам, то мы изменим алгоритм  $LU$ -разложения, как это показано на рис. 7.2. На  $k$ -м шаге изменённого алгоритма сначала формируется  $k$ -й столбец матрицы  $L$ ; это достигается векторной операцией деления. В самом внутреннем цикле (цикле по  $i$ )  $k$ -й столбец  $L$ , умноженный на число, вычитается из  $j$ -го столбца текущей матрицы  $A$ ; длина столбцов равна  $n - k$ . Таким образом, основной векторной операцией снова является триада, но теперь в качестве векторов выступают столбцы матрицы  $L$  и текущей матрицы  $A$ . В данный алгоритм также возможно внедрить любую из трёх стратегий выбора главного элемента (см. подразд. 3.2).

## 7.2 Распараллеливание вычислений

В этом разделе мы, следуя [9], излагаем некоторые понятия параллельных вычислений, которые в практике решения линейных систем имеют большое значение.

Параллельные вычисления реализуются не на обычных (скалярных) компьютерах, какими являются все персональные компьютеры, а на векторных или параллельных компьютерах. Их отличает то, что operandами команд компьютера являются не отдельные числовые величины (скаляры), а целые группы таких

величин, объединяемых в векторы или матрицы. Поэтому векторно-матричные операции для своего выполнения требуют вызова всего лишь одной команды. Выполнение таких команд начинается, как обычно, с загрузки операндов из памяти в векторно-матричный процессор и завершается обратным действием — записью результата операции в память. В промежутке между этими действиями операции реализуются на аппаратном уровне в процессоре.

**Векторные компьютеры.** В основе таких компьютеров лежит концепция *конвейеризации*. Это означает явное сегментирование процессора на отдельные части (сегменты), каждая из которых выполняет свою вычислительную подзадачу независимо для соответствующих частей operandов. Например, сумматор процессора для чисел с плавающей точкой разделён на шесть сегментов; каждый сегмент занят реализацией своей части операции сложения чисел. Всякий сегмент может работать только с одной парой operandов, а в целом на конвейере в текущий момент времени могут находиться шесть пар operandов. Преимущество подобной сегментации в том, что результаты выдаются в 6 раз быстрее (а в общем случае в  $K$ , где  $K$  — число сегментов сумматора), чем в скалярном процессоре, который, получив пару operandов, не принимает новой пары, пока не вычислит результат для первой пары. Для реализации этой возможности ускорения нужно подавать данные из памяти в процессор достаточно быстро, чтобы конвейер был всё время загружен данными и работой.

**Параллельные компьютеры.** В основе такого компьютера лежит идея использовать несколько процессоров, работающих сообща для решения одной задачи. Параллельный компьютер может иметь в своём составе либо очень простые процессоры, пригодные только для малых или ограниченных задач, либо набор полноценных процессоров, либо весьма мощные векторные процессоры. Все процессоры параллельного компьютера в каждый момент времени выполняют одну и ту же команду (или все простаивают) под управление главного процессора, называемого контроллером.

**Распараллеливание.** Независимо от того, на какой аппаратуре реализуется тот или иной вычислительный алгоритм, он обладает собственной, ему присущей характеристикой, показывающей возможности распараллеливания.

**Определение 7.2.** Степенью параллелизма численной задачи называется число её операций, которые можно выполнять параллельно.

**Пример 7.1.** Пусть требуется сложить два  $n$ -мерных вектора  $\mathbf{a}$  и  $\mathbf{b}$ . Сложение их элементов

$$a_i + b_i, \quad i = 1, \dots, n \quad (7.3)$$

независимы и потому могут выполняться параллельно. Степень параллелизма этого алгоритма равна  $n$ .

**Пример 7.2.** Пусть требуется сложить  $n$  чисел  $a_1, \dots, a_n$ . Обычный последовательный алгоритм

$$s := a_1, \quad s := s + a_i, \quad i = 1, \dots, n$$

не пригоден для параллельных вычислений. Однако в самой задаче заключен немалый параллелизм. Можно разбить операнды на «двойки», т. е., складывать их по двое на каждом этапе операции. Полностью эффект этой идеи проявляется, когда число операндов равно степени двойки, т. е.,  $n = 2^q$ . Если, например,  $q = 3$ , то всё сложение займёт  $q = 3$  этапа, на каждом этапе действия выполняются параллельно, как показано на рис. 7.3.

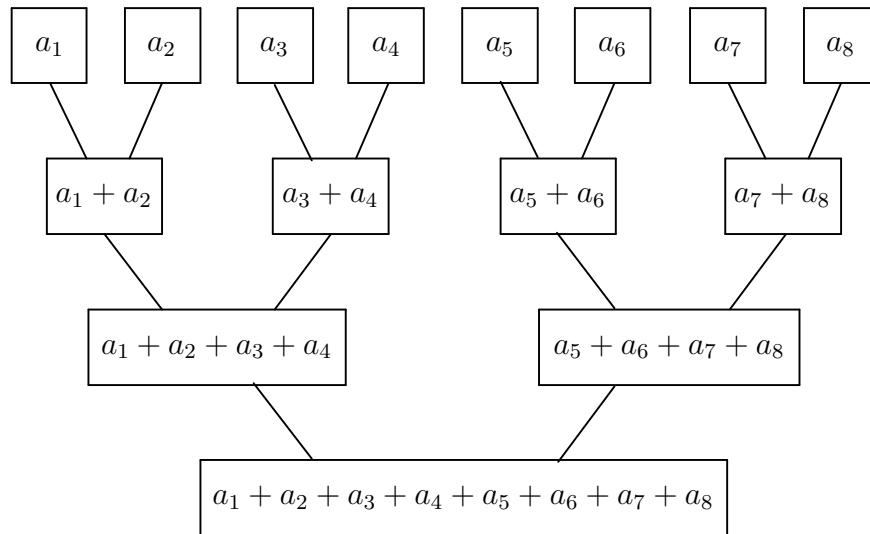


Рис. 7.3. Сложение  $n$  чисел методом сдвайвинга для  $n = 8$  [9]

Очевидно, на первом этапе степень параллелизма равна  $n/2$ , на втором  $n/4$  и т. д. В связи с этим приходим к обновлённому определению.

**Определение 7.3.** Средней степенью параллелизма численной задачи называется отношение общего числа операций её вычислительного алгоритма к числу последовательных этапов алгоритма.

Для приведённого примера 7.2 алгоритма сдваивания в задаче сложения  $n$  чисел средняя степень параллелизма равна

$$\frac{1}{q} \left( \frac{n}{2} + \frac{n}{4} + \cdots + 1 \right) = \frac{2^q - 1}{q} = \frac{n - 1}{\log n},$$

тогда как в предыдущем примере 7.1 средняя степень параллелизма максимальна. Этот алгоритм (7.3) обладает «идеальным» параллелизмом, в то время как для алгоритма на рис. 7.3 средняя степень параллелизма в  $\log n$  раз меньше идеальной.

### 7.3 Параллельное умножение матрицы на вектор

Пусть  $A$  – матрица размера  $(m \times n)$ , а  $\mathbf{x}$  – вектор длины  $n$ . Тогда

$$A\mathbf{x} = \begin{bmatrix} (\mathbf{a}_1, \mathbf{x}) \\ \cdots \\ (\mathbf{a}_m, \mathbf{x}) \end{bmatrix}, \quad (7.4)$$

где  $\mathbf{a}_i$  –  $i$ -я строка матрицы  $A$ , а  $(\mathbf{a}_i, \mathbf{x})$  – обычное скалярное произведение векторов  $\mathbf{a}_i$  и  $\mathbf{x}$ . Каждое из  $m$  имеющихся здесь скалярных произведений, как известно, требует суммирования  $n$  поэлементных произведений  $a_{ij}x_j$ . Как показано в предыдущем подразделе, такое суммирование можно распараллеливать сдваиванием, но такой параллелизм вычисления каждого отдельного скалярного произведения так или иначе неидеален. Однако  $m$  скалярных произведений в (7.4) можно вычислять параллельно. Другой способ умножения матрицы на вектор даётся формулой

$$A\mathbf{x} = \sum_{j=1}^n x_j \mathbf{a}_j, \quad (7.5)$$

где  $\mathbf{a}_j$  теперь обозначает  $j$ -й столбец матрицы  $A$ .

Различие представлений (7.4) и (7.5) можно рассматривать как различие двух способов доступа к данным в памяти, что показывают две программы на рис. 7.4. Программа слева на рис. 7.4 реализует метод (7.4), тогда как программа справа реализует метод (7.5), и различие здесь только в порядке индексов для циклов. Алгоритм, основанный на представлении (7.5), записывается так:

$$\mathbf{y} = \mathbf{0}, \quad \text{для } j \text{ от 1 до } n \text{ выполнить } \mathbf{y} = \mathbf{y} + x_j \mathbf{a}_j.$$

$y_i = 0$
Для $i = 1$ до $m$
Для $j = 1$ до $n$
$y_i = y_i + a_{ij}x_j$

$y_i = 0$
Для $j = 1$ до $n$
Для $i = 1$ до $m$
$y_i = y_i + a_{ij}x_j$

Рис. 7.4.  $ij$  (слева) и  $ji$  (справа) формы матрично-векторного умножения [9]

Как выше (в подразд. 7.1) говорилось, такая операция типа «вектор плюс произведение вектора на число», называется триадой (или операцией *saxpy*); некоторые векторные компьютеры выполняют её особенно эффективно.

Сравнение приведённых способов умножения матрицы на вектор показывает, что на одних векторных компьютерах предпочтителен один способ, на других – другой; многое определяется также и способом хранения данных в памяти. Предположим, что матрица  $A$  хранится по столбцам; такое соглашение в отношении двумерных массивов принято в Фортране. (В других языках, например в Паскале, двумерные массивы хранятся по строкам.) Тогда векторы, требуемые для алгоритма (7.5), располагаются в последовательно адресуемых ячейках памяти, в то время как для алгоритма (7.5) строки будут представлять векторы с шагом  $m$ . Однако в векторных компьютерах часто существует ограничение на доступ к векторам: в качестве operandов для векторных команд допускаются только векторы с шагом 1 (т. е. такие, которые располагаются в последовательно адресуемых ячейках памяти). Поэтому, если матрица  $A$  хранится по столбцам, то эти соображения, связанные с памятью, усиливают аргументацию в пользу алгоритма (7.5). Если же матрица  $A$  хранится по строкам, то предпочтительней может оказаться алгоритм (7.4). Только детальный анализ может показать, какой выбор следует сделать для конкретной машины.

## 7.4 Параллельное LU-разложение

Для распараллеленных вычислений нужны соответствующие параллельные или векторные компьютеры. С середины 70-х годов XX-го столетия фирма CRAY Research, Inc. производит векторные компьютеры, которые могут служить примером процессоров типа «регистр–регистр». Под этим подразумевается, что существуют векторные команды, для которых operandами являются векторы. Эти команды получают свои operandы из очень быстрой памяти, именуемой *векторными регистрами*, и запоминают результаты опять-таки в векторных регистрах. Для операции сложения векторов это показано на рис. 7.5.

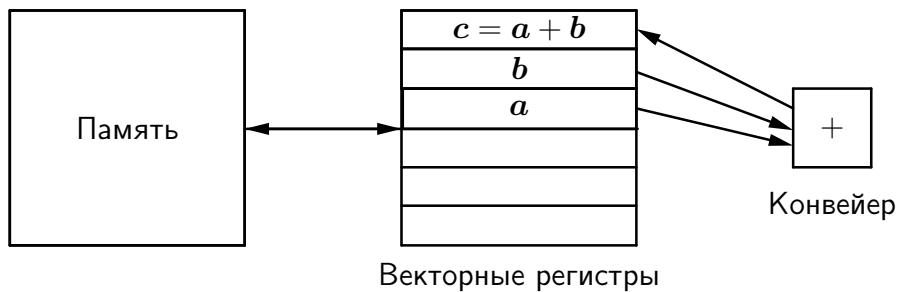


Рис. 7.5. Операция сложения в компьютере типа «регистр–регистр» [9]

Предполагается, что каждый векторный регистр состоит из некоторого числа слов. Например, в машинах CRAY имеется восемь векторных регистров, каждый ёмкостью в 64 числа с плавающей точкой. До начала сложения операнды загружаются из оперативной памяти в регистры. После завершения сложения векторный результат переписывается из регистровой памяти в оперативную память. Для таких машин желателен иной подход к организации *LU*-разложения.

Исследуем вначале характер обменов во внутреннем цикле столбцового алгоритма *LU*-разложения на рис. 7.2. Для простоты предположим, что столбец матрицы  $A$  полностью вкладывается в векторный регистр, и начнём с рассмотрения случая, когда на фоне вычислений может выполняться только загрузка или только запись в память. Несколько первых операций указаны в следующем списке:

Сформировать первый столбец матрицы  $L$ .

Загрузить второй столбец матрицы  $A$ .

$$\left\{ \begin{array}{l} \text{Модифицировать второй столбец матрицы } A; \\ \text{загрузить третий столбец матрицы } A. \end{array} \right. \quad (7.6)$$

Записать в память модифицированный второй столбец. (7.7)

$$\left\{ \begin{array}{l} \text{Модифицировать третий столбец матрицы } A; \\ \text{загрузить четвёртый столбец матрицы } A. \end{array} \right. \quad (7.8)$$

.....

Согласно (7.6), загрузка следующего столбца матрицы  $A$  совмещается с модификацией текущего столбца. Но затем возникает задержка при записи модифицированного второго столбца из регистра в память. Можно так модифицировать алгоритм, чтобы устранить задержку, вызванную записью в память (7.7). Идея состоит в том, чтобы выполнять необходимую обработку для  $j$ -го столбца полностью, прежде чем переходить к  $(j+1)$ -му столбцу. При

этом обработка каждого из остальных столбцов матрицы  $A$  откладывается до тех пор, пока не наступит время придать этому столбцу окончательный вид. Псевдокод данного алгоритма приведён на рис. 7.6.

```

Для  $j = 2$  до  $n$ 
    Для  $s = j$  до  $n$ 
         $l_{s,j-1} = a_{s,j-1}/a_{j-1,j-1}$ 
    Для  $k = 1$  до  $j - 1$ 
        Для  $i = k + 1$  до  $n$ 
             $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 

```

Рис. 7.6. Столбцово ориентированная схема  $\bar{L}U$ -разложения с отложенными модификациями ( $jk$ i-алгоритм, см. с. 328) [9]

Опишем несколько первых операций  $j$ -го шага вычислений, показывая таким образом характер обменов с памятью:

Загрузить первый столбец матрицы $L$ .	}
Загрузить $j$ -й столбец матрицы $A$ .	
Модифицировать $j$ -й столбец матрицы $A$ ;	}
загрузить второй столбец матрицы $L$ .	
Модифицировать $j$ -й столбец матрицы $A$ ;	}
загрузить третий столбец матрицы $L$ .	
.....	

(7.9)

Заметим, что в алгоритме (7.9) не производится записей в память, пока вся работа с  $j$ -м столбцом матрицы  $A$  не завершена. Столбцы матрицы  $L$  всё время должны загружаться в регистры, но эти загрузки идут на фоне вычислений. Только в начале и в конце каждого шага происходят задержки для загрузок и(или) записей. Вполне вероятно, что транслятор не сумеет распознать возможности оставить текущий  $j$ -й столбец в регистре; в этом случае результат, требуемый от алгоритма на рис. 7.6, либо достигается с переходом к программированию на языке ассемблера, либо аппроксимируется путём развёртывания циклов. Ещё одна потенциальная проблема при реализации данного алгоритма заключается в том, что длины векторов при модификациях непостоянны: на  $j$ -м шаге мы модифицируем  $j$ -й столбец, используя  $n - 1$  последних элементов столбца 1, далее  $n - 2$  последних элементов столбца 2 и т. д.

Алгоритм с отложенными модификациями не столь нужен для тех машин типа «регистр–регистр», в которых допускается совмещение с арифметикой как загрузок, так и записей в память. В этом случае операцию (7.7) можно было бы удалить, а операцию (7.8) заменить операцией

$$\left\{ \begin{array}{l} \text{Модифицировать третий столбец матрицы } A; \\ \text{загрузить четвёртый столбец матрицы } A; \\ \text{записать в память второй столбец матрицы } A. \end{array} \right.$$

Таким образом, запись в память второго столбца матрицы  $A$  происходит одновременно с загрузкой четвёртого столбца.

**Замечание 7.1.** Материал подразд. 7.2, 7.3 и 7.4 из [9] приведён, чтобы объяснить те приложения, для которых создаются алгоритмы с отложенными модификациями: алгоритм на рис. 7.6 и другие, показанные ниже в подразд. 7.5. Таким образом, включение таких алгоритмов в лабораторную работу (проект) может рассматриваться как задача имитирования алгоритмов векторных или параллельных компьютеров на обычных (скалярных) компьютерах с целью освоения этих современных версий  $LU$ -разложения.

## 7.5 $LU$ -разложение и его $ijk$ -формы

Ниже в описании  $ijk$ -алгоритмов факторизации (разложения), основанных на методе Гаусса исключения переменных, используем из [9] следующие обозначения для индексов:

- $k$  – номер исключаемой переменной,
- $i$  – номер строки, т. е., модифицируемого уравнения,
- $j$  – номер столбца, т. е., коэффициента в модифицируемом уравнении.

Тогда общую основу всех алгоритмов удобно определить тройкой вложенных циклов вида

Для_____
Для_____
Для_____
$a_{ij} = a_{ij} - l_{ik}a_{kj}$

Здесь последняя формула обозначает модификацию  $j$ -го элемента  $i$ -й строки матрицы  $A$  при исключении  $k$ -й переменной вектора неизвестных  $x$  из уравнений системы  $Ax = f$ . Перестановки трёх индексов для циклов определяют  $3! = 6$  возможных вариантов алгоритмов, образуя так называемые  $ijk$ -формы, для каждого вида разложения. Для квадратной матрицы  $A$  размера  $(n \times n)$  возможны четыре вида разложения, а именно:

$$A = L\bar{U}, A = \bar{L}U, A = U\bar{L}, A = \bar{U}L, \quad (7.10)$$

где черта сверху указывает на тот из сомножителей, который имеет единичную главную диагональ. Поэтому всего возможно построить 24 варианта  $ijk$ -алгоритмов разложения матрицы для решения различных задач: решения систем, обращения матрицы и вычисления её определителя.

Рассмотрим все шесть  $ijk$ -форм для одного из четырёх разложений (7.10), а именно, для  $\bar{L}U$ -разложения матрицы  $A$ . Для численной иллюстрации возьмём следующий пример.

### Пример 7.3.

$$A = \begin{bmatrix} 2 & 4 & -4 & 6 \\ 1 & 4 & 2 & 1 \\ 3 & 8 & 1 & 1 \\ 2 & 5 & 0 & 5 \end{bmatrix}, \bar{L} = \begin{bmatrix} 1 & & & \\ 1/2 & 1 & & \\ 3/2 & 1 & 1 & \\ 1 & 1/2 & 2/3 & 1 \end{bmatrix}, U = \begin{bmatrix} 2 & 4 & -4 & 6 \\ 2 & 4 & -2 & \\ 3 & -6 & & \\ 4 & & & \end{bmatrix}.$$

### Два алгоритма для $\bar{L}U$ -разложения матрицы $A$ с немедленными модификациями

#### 1) $kij$ -алгоритм, рис. 7.1.

Доступ к элементам матрицы  $A$  по строкам. Исключение по столбцам. Модификации немедленные. ГЭ – любая из трёх стратегий.

Для  $k = 1$  до  $n - 1$

Для  $i = k + 1$  до  $n$

$$l_{ik} = a_{ik}/a_{kk}$$

Для  $j = k + 1$  до  $n$

$$a_{ij} = a_{ij} - l_{ik}a_{kj}$$

#### 2) $kji$ -алгоритм, рис. 7.2.

Доступ к элементам матрицы  $A$  по столбцам. Исключение по столбцам. Модификации немедленные. ГЭ – любая из трёх стратегий.

Для  $k = 1$  до  $n - 1$

Для  $s = k + 1$  до  $n$

$$l_{sk} = a_{sk}/a_{kk}$$

Для  $j = k + 1$  до  $n$

Для  $i = k + 1$  до  $n$

$$a_{ij} = a_{ij} - l_{ik}a_{kj}$$

## Два алгоритма для $\bar{L}U$ -разложения матрицы $A$ (столбцово ориентированные с отложенными модификациями)

### 3) $jki$ -алгоритм, рис. 7.6.

Доступ к элементам матрицы  $A$  по столбцам. Исключение по столбцам. Модификации отложенные. ГЭ по  $(j-1)$ -му столбцу.

```

Для  $j = 2$  до  $n$ 
  Для  $s = j$  до  $n$ 
     $l_{s,j-1} = a_{s,j-1}/a_{j-1,j-1}$ 
  Для  $k = 1$  до  $j-1$ 
    Для  $i = k+1$  до  $n$ 
       $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
```

### 4) $jik$ -алгоритм.

Доступ к элементам матрицы  $A$  по столбцам. Исключение по столбцам. Модификации отложенные. В цикле по  $s$  идёт нормировка  $(j-1)$ -го столбца. Первый цикл по  $i$  вычисляет столбец для  $U$ , второй – столбец для  $L$ . ГЭ по  $(j-1)$ -му столбцу.

```

Для  $j = 2$  до  $n$ 
  Для  $s = j$  до  $n$ 
     $l_{s,j-1} = a_{s,j-1}/a_{j-1,j-1}$ 
  Для  $i = 2$  до  $j$ 
    Для  $k = 1$  до  $i-1$ 
       $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
  Для  $i = j+1$  до  $n$ 
    Для  $k = 1$  до  $j-1$ 
       $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
```

## Два алгоритма $ijk$ -форм для $\bar{L}U$ -разложения матрицы $A$ (строчно ориентированные с отложенными модификациями)

### 5) $ikj$ -алгоритм.

Доступ к элементам матрицы  $A$  по строкам. Исключение по строкам. Модификации отложенные. ГЭ по  $(i-1)$ -й строке.

```

Для  $i = 2$  до  $n$ 
  Для  $k = 1$  до  $i-1$ 
     $l_{i,k} = a_{i,k}/a_{k,k}$ 
  Для  $j = k+1$  до  $n$ 
     $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
```

### 6) $ijk$ -алгоритм.

Доступ к элементам матрицы  $A$  по строкам. Исключение по строкам. Модификации отложенные. Первый цикл по  $j$  находит элементы  $i$ -й строки  $L$ . Второй цикл по  $j$  – элементы  $i$ -й строки  $U$ . ГЭ по  $(i-1)$ -й строке.

```

Для  $i = 2$  до  $n$ 
  Для  $j = 2$  до  $i$ 
     $l_{i,j-1} = a_{i,j-1}/a_{j-1,j-1}$ 
    Для  $k = 1$  до  $j-1$ 
       $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
  Для  $j = i+1$  до  $n$ 
    Для  $k = 1$  до  $i-1$ 
       $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
```

**Замечание 7.2.** В приведённых алгоритмах не содержится процедура выбора главного элемента. Она дословно переносится из описания



Рис. 7.7. Способ доступа к данным для  $kij$ -формы (слева) и для  $kji$ -формы (справа)  $\bar{L}U$ -разложения. Обозначения:  $\bar{L}$ ,  $U$  – вычисление закончено, обращений больше нет;  $\boxtimes$  обозначает главный элемент (ГЭ);  $\emptyset$  – деление на ГЭ (нормировка) [9]

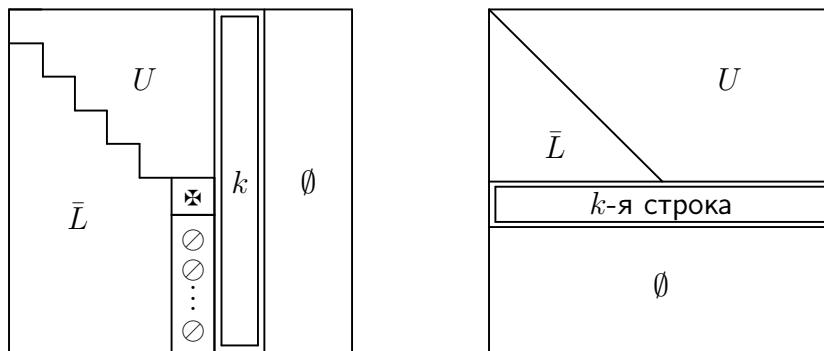


Рис. 7.8. Способ доступа к данным для  $jki$ -формы и для  $jik$ -формы (слева) и для  $ikj$ -формы и для  $ijk$ -формы (справа)  $\bar{L}U$ -разложения. Обозначения:  $\bar{L}$ ,  $U$  – вычисление закончено, обращения больше не производятся;  $\boxtimes$  обозначает главный элемент (ГЭ);  $\emptyset$  – деление на ГЭ (нормировка);  $\emptyset$  – обращений не было [9]

лабораторной работы № 1. Аналогичные алгоритмы могут быть написаны для остальных трёх видов разложения матрицы  $A$  из списка (7.10). При написании программ, соответствующих приведённым алгоритмам, следует выполнить требование, согласно которому все вычисления выполняются в одном и том же двумерном массиве, где сначала хранится матрица  $A$ . В процессе вычислений матрица  $A$  замещается элементами треугольных матриц, составляющих исковое разложение из списка (7.10). Способ доступа к данным для  $ijk$ -форм  $\bar{L}U$ -разложения показан на рис. 7.7 и рис. 7.8. Расчёты по алгоритмам  $kij$ -формы и  $kji$ -формы  $\bar{L}U$ -разложения достаточно очевидны. Для других четырёх форм  $\bar{L}U$ -разложения эти вычисления поясняются для примера 7.3 в табл. 7.1–7.4.

Таблица 7.1. Вычисления по алгоритму  $jki$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом

$A$	$j = 2$	$j = 3$	$j = 4$
<b>2</b> 4 -4 6	<b>2</b> 4 -4 6	<b>2</b> 4 -4 6	<b>2</b> 4 -4 6
1 4 2 1	1/2 <b>2</b> 2 1	1/2 <b>2</b> 4 1	1/2 <b>2</b> 4 -2
3 8 1 1	3/2 2 1 1	3/2 2/2 7 1	3/2 2/2 <b>3</b> -8
2 5 0 5	2/2 1 0 5	2/2 1/2 4 5	2/2 1/2 2/3 -1

$\uparrow\uparrow$ исходная матрица	$\uparrow\uparrow$ нормировка $(j-1)$ -го столбца; $(j-1)$ - кратное исключение в $j$ -м столбце	$\uparrow\uparrow$ нормировка $(j-1)$ -го столбца; $(j-1)$ - кратная модификация $j$ -го столбца	$\uparrow\uparrow$ нормировка $(j-1)$ -го столбца; $(j-1)$ - кратная модификация $j$ -го столбца
---	---	---	---

Таблица 7.2. Вычисления по алгоритму  $jik$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом

$A$	$j = 2$	$j = 3$	$j = 4$
<b>2</b> 4 -4 6	<b>2</b> 4 -4 6	<b>2</b> 4 -4 6	<b>2</b> 4 -4 6
1 4 2 1	1/2 <b>2</b> 2 1	1/2 <b>2</b> 4 1	1/2 <b>2</b> 4 -2
3 8 1 1	3/2 2 1 1	3/2 2/2 7 1	3/2 2/2 <b>3</b> -6
2 5 0 5	2/2 1 0 5	2/2 1/2 0 5	2/2 1/2 2/3 4

$\uparrow\uparrow$ исходная матрица	$\uparrow\uparrow$ нормировка $(j-1)$ -го столбца; $(j-1)$ - кратная модификация $j$ -го столбца	$\uparrow\uparrow$ нормировка $(j-1)$ -го столбца; $(j-1)$ - кратная модификация $j$ -го столбца
---	---	---

Таблица 7.3. Вычисления по алгоритму  $ikj$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом

$A$	$i = 2$	$i = 3$	$i = 4$
$\begin{matrix} 2 & 4 & -4 & 6 \\ 1 & 4 & 2 & 1 \\ 3 & 8 & 1 & 1 \\ 2 & 5 & 0 & 5 \end{matrix}$	$\begin{matrix} \mathbf{2} & 4 & -4 & 6 \\ 1/2 & 2 & 4 & -2 \\ 3 & 8 & 1 & 1 \\ 2 & 5 & 0 & 5 \end{matrix}$	$\begin{matrix} \mathbf{2} & 4 & -4 & 6 \\ 1/2 & 2 & 4 & -2 \\ 3/2 & 2 & 7 & -8 \\ 2 & 5 & 0 & 5 \end{matrix}$	$\begin{matrix} \mathbf{2} & 4 & -4 & 6 \\ 1/2 & \mathbf{2} & 4 & -2 \\ 3/2 & 2/2 & 3 & -6 \\ 2/2 & 1 & 4 & -1 \end{matrix}$
$\uparrow\uparrow$ исходная матрица	$\uparrow\uparrow$ $(i-1)$ нормировок и вычитаний в $i$ -й строке	$\uparrow\uparrow$ $(i-1)$ нормировок и вычитаний в $i$ -й строке	$\uparrow\uparrow$ $(i-1)$ нормировок и вычитаний в $i$ -й строке

## 7.6 Треугольные системы

По окончании этапа приведения в гауссовом исключении нам необходимо решить треугольную систему уравнений

$$\left[ \begin{array}{cccc} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{22} & \cdots & u_{2n} & \\ \ddots & \vdots & & \\ u_{nn} & & & \end{array} \right] \left[ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right] = \left[ \begin{array}{c} c_1 \\ c_2 \\ \vdots \\ c_n \end{array} \right].$$

Обычный алгоритм обратной подстановки описывается формулами

$$x_i = (c_i - u_{i,i+1}x_{i+1} - \dots - u_{in}x_n)/u_{ii}, \quad i = n, \dots, 1. \quad (7.11)$$

Рассмотрим, как он может быть реализован в векторных операциях. Если  $U$  хранится по строкам (так будет, если на этапе приведения  $A$  хранилась по строкам), то формулы (7.11) задают скалярные произведения с длинами векторов, меняющимися от 1 до  $n-1$ , и  $n$  скалярных делений (рис. 7.9 слева).

Альтернативный алгоритм, полезный, если  $U$  хранится по столбцам, представлен в виде псевдокода на рис. 7.9 справа. Он называется *столбцовыми алгоритмом* (или *алгоритмом векторных сумм*).

Таблица 7.4. Вычисления по алгоритму  $ijk$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом

$A$
2 4 -4 6
1 4 2 1
3 8 1 1
2 5 0 5

$i = 2, j = 2, k = 1$
<b>2</b> 4 -4 6
1/2 2 2 1
3 8 1 1
2 5 0 5

$i = 3, j = 2, k = 1$
<b>2</b> 4 -4 6
1/2 2 4 -2
3/2 2 1 1
2 5 0 5

$i = 4, j = 2, k = 1$
<b>2</b> 4 -4 6
1/2 2 4 -2
3/2 2/2 3 -6
2/2 1 0 5

↑  
исходная  
матрица

$i = 2, j = 3, k = 1$
<b>2</b> 4 -4 6
1/2 2 4 1
3 8 1 1
2 5 0 5

$i = 3, j = 3, k = 1$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 7 1
2 5 0 5

$i = 4, j = 3, k = 1$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 3 -6
2/2 1/2 4 5

$i = 2, j = 4, k = 1$
<b>2</b> 4 -4 6
1/2 2 4 -2
3 8 1 1
2 5 0 5

$i = 3, j = 3, k = 2$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 3 1
2 5 0 5

$i = 4, j = 3, k = 2$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 3 -6
2/2 1/2 2 5

↑  
( $i - 1$ )  
нормировок  
в  $i$ -й строке

$i = 3, j = 4, k = 1$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 3 -8
2 5 0 5

$i = 4, j = 4, k = 1$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 <b>3</b> -6
2/2 1/2 2/3 -1

↑  
 $(n - i)(i - 1) +$   
 $\frac{i(i - 1)}{2}$   
вычитаний  
в  $i$ -й строке

$i = 3, j = 4, k = 2$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 3 -6
2 5 0 5

$i = 4, j = 4, k = 2$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 3 -6
2/2 1/2 2/3 0

↑  
( $i - 1$ )  
нормировок и  
 $(n - i)(i - 1) +$   
 $\frac{i(i - 1)}{2}$   
вычитаний  
в  $i$ -й строке

$i = 4, j = 4, k = 3$
<b>2</b> 4 -4 6
1/2 <b>2</b> 4 -2
3/2 2/2 <b>3</b> -6
2/2 1/2 2/3 4

Для $j = n$ до 1 с шагом $-1$ Для $j = i + 1$ до $n$ $c_i = c_i - u_{ij}x_j$ $x_i = c_i/u_{ii}$	Для $j = n$ до 1 с шагом $-1$ $x_j = c_j/u_{jj}$ Для $i = j - 1$ до 1 с шагом $-1$ $c_i = c_i - x_j u_{ij}$
--	--

Рис. 7.9. Алгоритмы скалярных произведений (слева) и столбцовый для обратной подстановки (справа)

Как только найдено значение  $x_n$ , вычисляются и вычтываются из соответствующих элементов  $c_i$  величины произведений  $x_n u_{in}$  ( $i=1, \dots, n-1$ ); таким образом, вклад, вносимый  $x_n$  в прочие компоненты решения, реализуется до перехода к следующему шагу. Шаг с номером  $i$  состоит из скалярного деления, сопровождаемого триадой длины  $i-1$  (подразумевается, что шаги нумеруются в обратном порядке:  $n, n-1, \dots, 2, 1$ ). Какой из двух алгоритмов выбрать, диктуется способом хранения матрицы  $U$ , если он был определён  $LU$ -разложением.

Как алгоритм скалярных произведений, так и столбцовый алгоритм легко переформулировать на случай нижнетреугольных систем, процесс решения которых называется алгоритмом *прямой подстановки*.

## 7.7 Задание на лабораторный проект № 4

Написать и отладить программу, реализующую ваш вариант метода исключения с выбором главного элемента, для численного решения систем линейных алгебраических уравнений  $Ax = f$ , вычисления  $\det A$  и  $A^{-1}$ . Предусмотреть сообщения, предупреждающие о невозможности решения указанных задач с заданной матрицей  $A$ .

Отделить следующие основные части программы:

- а) подпрограмму факторизации матрицы  $A$ , отвечающую вашему варианту метода исключения;
- б) подпрограмму решения систем линейных алгебраических уравнений;
- в) подпрограмму вычисления определителя матриц;
- г) подпрограммы обращения матриц;
- д) сервисные подпрограммы.

Уделить особое внимание эффективности программы (в смысле экономии оперативной памяти). Предусмотреть пошаговое выполнение алгоритма исключения с выводом результата на экран.

Выполнить следующие пункты задания.

1. Провести подсчёт фактического числа выполняемых операций умножения и деления при решении системы линейных алгебраических уравнений, сравнить его с оценочным числом ( $n^3/3$ ).

2. Определить скорость решения задач (решение систем линейных алгебраических уравнений, обращение матриц) с учётом времени, затрачиваемого на разложение матрицы. Для этого спроектировать и провести эксперимент, который охватывает матрицы порядка от 5 до 100 (через 5 порядков). Представить результаты в виде таблицы и графика зависимости времени выполнения (в минутах и секундах) от порядка матриц. Таблицу и график вывести на экран.

3. Оценить точность решения систем линейных алгебраических уравнений, имеющих тот же самый порядок, что и задачи из пункта 2. Для этого сгенерировать случайные матрицы  $A$ , задать точное решение  $x^*$  и образовать правые части  $f = Ax^*$ . Провести анализ точности вычисленного решения  $x$  от порядка матрицы. Результаты представить в виде таблицы и графика.

Для заполнения матрицы  $A$  использовать случайные числа из диапазона от  $-100$  до  $100$ . В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – порядок матрицы. Для оценки точности использовать норму вектора

$$\|x\|_\infty = \max_i(|x_i|). \quad (7.12)$$

4. Повторить пункт 3 задания для плохо обусловленных матриц (см. подразд. 3.6), имеющих порядок от 4 до 40 с шагом 4.

5. Вычислить матрицу  $A^{-1}$  следующими двумя способами.

Способ 1 – через решение системы  $AX = I$ , где  $I$  – единичная матрица.

Способ 2 – через разложение матрицы  $A$  в произведение элементарных матриц, обращение которых осуществляется аналитически, а их произведение даёт матрицу  $A^{-1}$ .

Сравнить затраты машинного времени и точность обращения матриц при использовании указанных способов 1 и 2. Эксперименты провести для случайных матриц порядков от 5 до 100 через 5. Для оценки точности в обоих способах использовать оценочную формулу

$$\|A_{\text{т}}^{-1} - A_{\text{пр}}^{-1}\| \leq \|I - AA_{\text{пр}}^{-1}\| \cdot \|A\|^{-1}. \quad (7.13)$$

В выражении (7.13), где  $A_{\text{т}}^{-1}$  – точное значение обратной матрицы, а  $A_{\text{пр}}^{-1}$  – приближенное значение, полученное в результате обращения каждым из способов 1 и 2, норму матрицы вычислять в соответствии со следующим определением:

$$\|A\|_{\infty} = \max_i \left( \sum_{j=1}^n |a_{ij}| \right). \quad (7.14)$$

6. Провести подсчёт фактического числа выполняемых операций умножения и деления при обращении матриц первым и вторым способами, сравнить его с оценочным числом ( $n^3$ ).

**Замечание 7.3.** По ходу проведения численных экспериментов на экран дисплея должны выводиться следующие таблицы.

#### Решение систем линейных алгебраических уравнений:

Порядок	Время	Точность	Теоретическое число операций	Реальное число операций
---------	-------	----------	------------------------------	-------------------------

Аналогичная таблица должна быть построена для плохо обусловленных матриц.

#### Обращение матриц:

Порядок	Время		Точность		Число операций		
	спос. 1	спос. 2	спос. 1	спос. 2	спос. 1	спос. 2	теорет.

**Замечание 7.4.** Результаты экспериментов необходимо вывести на экран в форме следующих графиков.

#### Графики решения систем линейных алгебраических уравнений:

- зависимость реального и расчётного числа операций от порядка матрицы (для разных графиков использовать разные цвета);
- зависимости времени и точности решения от порядка матриц.

#### Графики для обращения матриц:

- зависимость реального и оценочного числа операций от порядка матрицы (для разных графиков использовать разные цвета);
- зависимости времени и точности обращения первым и вторым способом от порядка матриц.

Таблица 7.5. Варианты задания на лабораторный проект № 4

Вид разложения	<i>kij</i>			<i>kji</i>			<i>jki</i>	<i>jik</i>	<i>ikj</i>	<i>ijk</i>
	a	b	c	a	b	c	a	a	b	b
$A = \bar{L}U$	1	2	3	4	5	6	7	8	9	10
$A = L\bar{U}$	11	12	13	14	15	16	17	18	19	20
$A = \bar{U}L$	21	22	23	24	25	26	27	28	29	30
$A = U\bar{L}$	31	32	33	34	35	36	37	38	39	40

<sup>a</sup> – выбор ГЭ по столбцу активной подматрицы;

<sup>b</sup> – выбор ГЭ по строке активной подматрицы;

<sup>c</sup> – выбор ГЭ по активной подматрице.

## 7.8 Варианты задания на лабораторный проект № 4

В табл. 7.5 приведены 40 номеров вариантов задания на лабораторную работу (проект) № 4 с тремя стратегиями выбора главного элемента.

## 7.9 Тестовые задачи для проекта № 4

Используйте приводимые ниже задачи в двух режимах:

- для контроля собственного понимания алгоритма,
- для контроля правильности вашего программирования.

### Задача 1

Для матрицы

$$A = \begin{pmatrix} 2 & 0 & 2 \\ 4 & -1 & 3 \\ -2 & -3 & -2 \end{pmatrix}$$

выполнить следующее:

- Построить  $\bar{L}U$ -разложение матрицы  $A$  ( $\bar{L}$  с единицами на диагонали).
- С помощью  $\bar{L}U$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, 0, -3)^T$ .

- в. С помощью  $\overline{LU}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 2

Для матрицы

$$A = \begin{pmatrix} 3 & 6 & 2 \\ -5 & -10 & -4 \\ 1 & 3 & 1 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $\overline{U}L$ -разложение матрицы  $A$  ( $\overline{U}$  с единицами на диагонали).
- б. С помощью  $\overline{U}L$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (10, -16, 5)^T$ .

- в. С помощью  $\overline{U}L$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 3

Для матрицы

$$A = \begin{pmatrix} 3 & 0 & 3 \\ -1 & 1 & -2 \\ 1 & 2 & 1 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $L\overline{U}$ -разложение матрицы  $A$  ( $\overline{U}$  с единицами на диагонали).
- б. С помощью  $L\overline{U}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, -2, -2)^T$ .

- в. С помощью  $L\overline{U}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 4**

Для матрицы

$$A = \begin{pmatrix} -3 & 1 & 1 \\ 2 & 1 & 2 \\ 4 & 0 & 2 \end{pmatrix}$$

выполнить следующее:

- а. Построить  $U\bar{L}$ -разложение матрицы  $A$  ( $\bar{L}$  с единицами на диагонали).
- б. С помощью  $U\bar{L}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (5, 2, 0)^T$ .

- в. С помощью  $U\bar{L}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

## 7.10 Заключение по разделу 7

В данном разделе рассмотрены стандартные алгоритмы  $LU$ -разложения матрицы, численно эквивалентные методу Гаусса последовательного исключения неизвестных в системе линейных алгебраических уравнений. Даны теоретические сведения, достаточные для выполнения лабораторного проекта № 4.

Проект № 4 отличается от проекта № 1 тем, что в нём предлагается выполнить программирование  $LU$ -разложения по тем алгоритмам, которые приспособлены для реализации в векторных или матричных процессорах, когда одной командой такого процессора выполняется операция сразу над всеми элементами вектора. Эти алгоритмы существенно привязаны в способу доступа к элементам матрицы – по столбцам или по строкам.

В проекте на этой основе решаются две основные задачи вычислительной линейной алгебры: (1) найти решение СЛАУ и (2) найти обратную матрицу. При этом матрицу системы предлагается формировать тремя различными способами: вводить «вручную» с клавиатуры компьютера, генерировать случайным образом или же из списка специальных – плохо обусловленных – матриц.

Этот проект мы рассматриваем как (дополнительный) проект повышенной сложности (по сравнению с проектом № 1) для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

# 8

## Проект № 5 «Алгоритмы окаймления в LU-разложении»

### 8.1 Метод окаймления

Хотя  $ijk$ -формы дают шесть различных способов организации  $LU$ -разложения, имеются и другие способы, потенциально полезные для векторных компьютеров. Даже тогда, когда та или иная  $ijk$ -форма теоретически пригодна для конкретной векторной машины, при её реализации могут возникнуть проблемы, особенно если применяется язык высокого уровня. Разбираемые ниже способы организации вычислений основаны на операциях с подматрицами; потенциально они проще реализуются и облегчают написание переносимых программ.

В основе этих способов организации лежит идея **окаймления** [15]. Математически её можно представить следующим образом. Разобьём матрицу  $A$  на блоки и, соответственно, разобьём на блоки сомножители  $\bar{L}$  и  $U$  искомого разложения  $\bar{L}U = A$ :

$$\begin{bmatrix} \bar{L}_{11} & 0 & 0 \\ \mathbf{l}_{j1}^T & 1 & 0 \\ L_{31} & \mathbf{l}_{3j} & \bar{L}_{33} \end{bmatrix} \begin{bmatrix} U_{11} & \mathbf{u}_{1j} & U_{13} \\ 0 & u_{jj} & \mathbf{u}_{j3}^T \\ 0 & 0 & U_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & \mathbf{a}_{1j} & A_{13} \\ \mathbf{a}_{j1}^T & a_{jj} & \mathbf{a}_{j3}^T \\ A_{31} & \mathbf{a}_{3j} & A_{33} \end{bmatrix}. \quad (8.1)$$

Здесь  $\mathbf{l}_{j1}^T$ ,  $\mathbf{u}_{j3}^T$ ,  $\mathbf{a}_{j1}^T$  и  $\mathbf{a}_{j3}^T$  – векторы-строки, а  $\mathbf{l}_{3j}$ ,  $\mathbf{u}_{1j}$ ,  $\mathbf{a}_{1j}$  и  $\mathbf{a}_{3j}$  – векторы-столбцы; каждый из этих элементов находится на  $j$ -й позиции.

### 8.2 Окаймление известной части разложения

Пусть известно разложение  $\bar{L}_{11}U_{11} = A_{11}$ , которое можно рассматривать как равенство (8.1) для блока  $A_{11}$ . Запишем аналогичные равенства для тех трёх

блоков, которые окаймляют эту известную часть разложения, следуя правилу перемножения блок-матриц, а именно, для  $\mathbf{a}_{j1}^T$ ,  $\mathbf{a}_{1j}$  и  $a_{jj}$ :

$$\mathbf{l}_{j1}^T \mathbf{U}_{11} = \mathbf{a}_{j1}^T, \quad \overline{\mathbf{L}}_{11} \mathbf{u}_{1j} = \mathbf{a}_{1j}, \quad \mathbf{l}_{j1}^T \mathbf{u}_{1j} + u_{jj} = a_{jj}. \quad (8.2)$$

Из первого уравнения (8.2), переписанного в виде  $\mathbf{U}_{11}^T \mathbf{l}_{j1} = \mathbf{a}_{j1}$ , находим  $\mathbf{l}_{j1}$ , из второго находим  $\mathbf{u}_{1j}$  и затем из третьего находим  $u_{jj} = a_{jj} - \mathbf{l}_{j1}^T \mathbf{u}_{1j}$ . При этом первое и второе уравнения описываются следующими нижнетреугольными системами

$$\mathbf{U}_{11}^T \mathbf{l}_{j1} = \mathbf{a}_{j1}, \quad \overline{\mathbf{L}}_{11} \mathbf{u}_{1j} = \mathbf{a}_{1j}. \quad (8.3)$$

Существуют два естественных варианта реализации окаймления известной части  $LU$ -разложения.

В первом варианте треугольные системы (8.3) решаются с помощью столбцового алгоритма, во втором – с помощью алгоритма скалярных произведений. Псевдокоды этих двух вариантов приведены на рис. 8.1.

<pre> Для j = 2 до n     Для k = 1 до j - 2         Для i = k + 1 до j - 1             aij = aij - lik akj         Для k = 1 до j - 1             ljk = ajk /akk         Для i = k + 1 до j             aji = aji - ljk aki     </pre>	<pre> Для j = 2 до n     Для i = 2 до j - 1         Для k = 1 до i - 1             aij = aij - lik akj         Для i = 2 до j             lji-1 = aji-1 / aii-1,i-1         Для k = 1 до i - 1             aji = aji - lji-1 aki     </pre>
--	---

Рис. 8.1. Алгоритмы окаймления известной части  $\overline{\mathbf{L}}\mathbf{U}$ -разложения: столбцовый (слева) и алгоритм скалярных произведений (справа)

В первом цикле по  $i$  на рис. 8.1 (слева) выполняется модификация  $j$ -го столбца матрицы  $A$  и тем самым вычисляется  $j$ -й столбец матрицы  $U$ . Во втором цикле по  $i$  модифицируется  $j$ -я строка матрицы  $A$  и вычисляется  $j$ -я строка матрицы  $\overline{\mathbf{L}}$ . Заметим, что при  $i = j$  во втором цикле по  $i$  пересчитывается элемент  $(j, j)$  матрицы  $A$ ; в результате, согласно (8.2), получается  $u_{jj}$ .

Во второй форме алгоритма окаймления на рис. 8.1 (справа) первый цикл по  $i, k$  вычисляет  $j$ -й столбец матрицы  $U$ , для чего из элементов  $a_{ij}$  вычтываются скалярные произведения строк с 2-й по  $(j - 1)$ -ю матрицы  $\overline{\mathbf{L}}$  с  $j$ -м столбцом  $U$ . Это эквивалентно решению системы  $\overline{\mathbf{L}}_{11} \mathbf{u}_{1j} = \mathbf{a}_{1j}$ . Во втором цикле по

$i, k$  модифицируется  $j$ -я строка  $A$  путём делений (нормировок) элементов этой строки, сопровождаемых вычитаниями скалярных произведений  $j$ -й строки  $\bar{L}$  и столбцов  $U$ . Это эквивалентно решению треугольной системы  $U_{11}^T \mathbf{l}_{j1} = \mathbf{a}_{j1}$  относительно  $j$ -й строки матрицы  $\bar{L}$ . Отметим, что здесь при  $j = i$  модифицируется элемент  $(j, j)$  матрицы  $A$ , и это относится уже к вычислению  $j$ -го столбца матрицы  $U$ ; в результате получается элемент  $u_{jj}$ . Вычисления по этим формам показаны в табл. 8.1.

Таблица 8.1. Вычисления по алгоритмам на рис. 8.1 для примера 7.3. Позиции элемента-делителя столбца  $\bar{L}$  показаны выделенным шрифтом

$A$	$j = 2$	$j = 3$	$j = 4$
<b>2</b> 4 -4 6 1 4 2 1 3 8 1 1 2 5 0 5	<b>2</b> 4 -4 6 $1/2$ <b>2</b> 2 1 3 8 1 1 2 5 0 5	<b>2</b> 4 -4 6 $1/2$ <b>2</b> 4 1 $3/2$ $2/2$ <b>3</b> 1 2 5 0 5	<b>2</b> 4 -4 6 $1/2$ <b>2</b> 4 -2 $3/2$ $2/2$ <b>3</b> -6 $2/2$ $1/2$ $2/3$ 4

В обеих формах окаймления обращения к данным производятся одинаково, что показано на рис. 8.2.

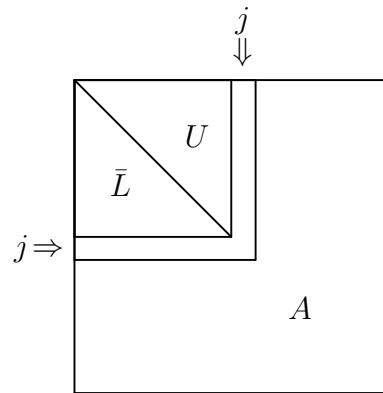


Рис. 8.2. Доступ к данным в алгоритмах окаймления известной части разложения.  $\bar{L}$ ,  $U$  здесь вычисление закончено, но обращения происходят.  $A$  – обращений не было. Вычисляются:  $j$ -й столбец матрицы  $U$  и  $j$ -я строка матрицы  $\bar{L}$

Обратим внимание, что в обоих случаях требуется доступ и к строкам, и к столбцам матрицы  $A$ . Поэтому алгоритмы будут неэффективны для векторных компьютеров, требующих, чтобы элементы вектора находились в смежных позициях памяти. Ещё одной сложностью является внедрение процедуры выбора главного элемента в эти алгоритмы.

### 8.3 Окаймление неизвестной части разложения

Основная работа в алгоритмах окаймления приходится на решение треугольных систем (8.3). Это матрично-векторные операции, которые можно реализовать в виде подпрограмм по типу рис. 8.1, добиваясь в них максимальной для данной машины эффективности. Ещё один способ организации вычислений, который называют алгоритмом Донгарры–Айзенштата, имеет то преимущество, что его основной операцией является матрично-векторное умножение. Математически алгоритм можно описать следующим образом. Выпишем из равенства (8.1) три других соотношения, на этот раз для  $a_{jj}$ ,  $\mathbf{a}_{3j}$  и  $\mathbf{a}_{j3}^T$ . Отсюда получим

$$u_{jj} = a_{jj} - \mathbf{l}_{j1}^T \mathbf{u}_{1j}, \quad \mathbf{u}_{j3}^T = \mathbf{a}_{j3}^T - \mathbf{l}_{j1}^T U_{13}, \quad \mathbf{l}_{3j} = (\mathbf{a}_{3j} - L_{31} \mathbf{u}_{1j}) / u_{jj}. \quad (8.4)$$

Характер доступа к данным при таком вычислении показан на рис. 8.3.

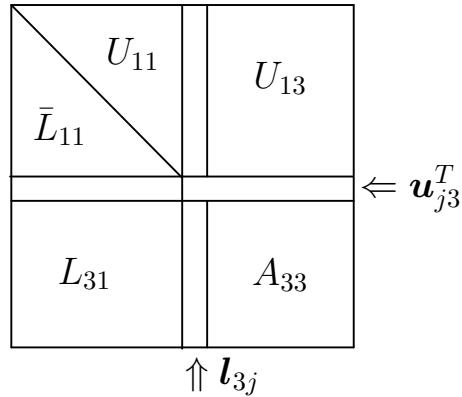


Рис. 8.3. Доступ к данным в алгоритмах окаймления неизвестной части разложения.  $\bar{L}_{11}$ ,  $U_{11}$  – вычисление закончено, обращений больше нет.  $L_{31}$ ,  $U_{13}$  здесь вычисление закончено, но обращения происходят.  $A_{33}$  – обращений не было. Вычисляются:  $j$ -й столбец  $\mathbf{l}_{3j}$  матрицы  $\bar{L}$  и  $j$ -я строка  $\mathbf{u}_{j3}^T$  матрицы  $U$

Видно, что блоки  $U_{13}$  и  $L_{31}$ , необходимые для вычислений (8.4), на каждый такой момент времени уже известны, так же как и все другие величины в правых частях равенств (8.4), поэтому здесь нет решения уравнений.

Основной операцией в (8.4) является умножение вектора на прямоугольную матрицу. Можно реализовать такие умножения посредством скалярных произведений или линейных комбинаций, что приводит к двум различным формам алгоритма, показанным на рис. 8.4.

Для $j = 1$ до $n$ Для $k = 1$ до $j - 1$ Для $i = j$ до $n$ $a_{ji} = a_{ji} - l_{jk}a_{ki}$ Для $k = 1$ до $j - 1$ Для $i = j + 1$ до $n$ $a_{ij} = a_{ij} - l_{ik}a_{kj}$ Для $s = j + 1$ до $n$ $l_{sj} = a_{sj}/a_{jj}$	Для $j = 1$ до $n$ Для $i = j + 1$ до $n$ Для $k = 1$ до $j - 1$ $a_{ij} = a_{ij} - l_{ik}a_{kj}$ Для $i = j$ до $n$ Для $k = 1$ до $j - 1$ $a_{ji} = a_{ji} - l_{jk}a_{ki}$ Для $s = j + 1$ до $n$ $l_{sj} = a_{sj}/a_{jj}$
--	--

Рис. 8.4. Алгоритмы Донгарры–Айзенштата окаймления неизвестной части  $\bar{L}U$ -разложения: алгоритм линейных комбинаций (слева) и алгоритм скалярных произведений (справа)

Первый цикл по  $k,i$  на рис. 8.4 (слева) производит последовательные модификации  $j$ -й строки матрицы  $A$ , которая по окончании цикла  $k$  превращается в  $j$ -ю строку матрицы  $U$ . Эти модификации можно рассматривать как вычитание векторно-матричного произведения  $\mathbf{l}_{j1}^T U_{13}$  из  $\mathbf{a}_{j3}^T$  во второй формуле  $\mathbf{u}_{j3}^T = \mathbf{a}_{j3}^T - \mathbf{l}_{j1}^T U_{13}$  в (8.4) с помощью линейных комбинаций строк  $U$ . В случае  $j = i$  результатом модификации будет первая величина  $u_{jj}$  в (8.4). Во второй паре циклов по  $k$  и  $i$  выполняются модификации  $j$ -го столбца матрицы  $A$  по формуле  $\mathbf{l}_{3j} = (\mathbf{a}_{3j} - L_{31}\mathbf{u}_{1j})$ , т. е., по второй формуле (8.4) с точностью до деления на  $u_{jj}$  с вычислением матрично-векторного произведения  $L_{31}\mathbf{u}_{1j}$  в (8.4) посредством линейных операций столбцов матрицы  $L$ . Обратите внимание, — в отличие от алгоритма отложенных модификаций на рис. 7.6, теперь длины векторов, участвующих в линейных комбинациях, одинаковы. Второй оператор цикла с индексом  $k$  можно удалить, причём программа снова будет верна; мы вставили этот оператор, чтобы подчеркнуть наличие линейных комбинаций. Отметим ещё, что в первом цикле по  $k,i$  происходит обращение к строкам матрицы  $A$ , а во втором цикле по  $k,i$  — к её столбцам. Следовательно, эта форма неэффективна для векторных компьютеров, требующих размещения элементов вектора в смежных ячейках памяти.

Алгоритм на рис. 8.4 (справа) использует скалярные произведения. На  $j$ -м шаге первый цикл по  $i, k$  вычисляет, с точностью до финального деления,  $j$ -й столбец матрицы  $L$ ; с этой целью  $j$ -й столбец в  $A$  модифицируется посредством скалярных произведений строк  $L$  и  $j$ -го столбца  $U$ . Во втором цикле по  $i, k$  вычисляется  $j$ -я строка матрицы  $U$ , для чего  $j$ -я строка в  $A$  модифициру-

ется посредством скалярных произведений  $j$ -й строки  $L$  и столбцов  $U$ . Снова требуется доступ к строкам и столбцам матрицы  $A$ . Потенциальное преимущество алгоритма Донгарры–Айзенштата заключается в том, что в некоторых векторных компьютерах матрично-векторные умножения выполняются весьма эффективно. Вычисления по этим формам показаны в табл. 8.2.

Таблица 8.2. Вычисления по алгоритмам на рис. 8.4 для примера 7.3. Позиции элемента–делителя столбца  $\overline{L}$  показаны выделенным шрифтом

$j = 1$	$j = 2$	$j = 3$	$j = 4$
<b>2</b> 4 -4 6	<b>2</b> 4 -4 6	<b>2</b> 4 -4 6	<b>2</b> 4 -4 6
1/2 4 2 1	1/2 <b>2</b> 4 -2	1/2 <b>2</b> 4 -2	1/2 <b>2</b> 4 -2
3/2 8 1 1	3/2 2/2 1 1	3/2 2/2 <b>3</b> -6	3/2 2/2 <b>3</b> -6
2/2 5 0 5	2/2 1/2 0 5	2/2 1/2 2/3 5	2/2 1/2 2/3 4

Как и в алгоритмах окаймления известной части разложения (подразд. 8.2), в данных алгоритмах дополнительную сложность представляет внедрение процедуры выбора главного элемента.

## 8.4 Задание на лабораторный проект № 5

Написать и отладить программу, реализующую ваш вариант метода исключения с выбором главного элемента, для численного решения систем линейных алгебраических уравнений  $Ax = f$ , вычисления  $\det A$  и  $A^{-1}$ . Предусмотреть сообщения, предупреждающие о невозможности решения указанных задач с заданной матрицей  $A$ .

Отделить следующие основные части программы:

- а) подпрограмму факторизации матрицы  $A$ , отвечающую вашему варианту метода исключения;
- б) подпрограмму решения систем линейных алгебраических уравнений;
- в) подпрограмму вычисления определителя матриц;
- г) подпрограммы обращения матриц;
- д) сервисные подпрограммы.

Уделить особое внимание эффективности программы (в смысле экономии оперативной памяти). Предусмотреть пошаговое выполнение алгоритма исключения с выводом результата на экран.

Выполнить следующие пункты задания.

1. Провести подсчёт фактического числа выполняемых операций умножения и деления при решении системы линейных алгебраических уравнений, сравнить его с оценочным числом ( $n^3/3$ ).

2. Определить скорость решения задач (решение систем линейных алгебраических уравнений, обращение матриц) с учётом времени, затрачиваемого на разложение матрицы. Для этого спроектировать и провести эксперимент, который охватывает матрицы порядка от 5 до 100 (через 5 порядков). Представить результаты в виде таблицы и графика зависимости времени выполнения (в минутах и секундах) от порядка матриц. Таблицу и график вывести на экран.

3. Оценить точность решения систем линейных алгебраических уравнений, имеющих тот же самый порядок, что и задачи из пункта 2. Для этого сгенерировать случайные матрицы  $A$ , задать точное решение  $x^*$  и образовать правые части  $f = Ax^*$ . Провести анализ точности вычисленного решения  $x$  от порядка матрицы. Результаты представить в виде таблицы и графика.

Для заполнения матрицы  $A$  использовать случайные числа из диапазона от  $-100$  до  $100$ . В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)^T$ , где  $n$  – порядок матрицы. Для оценки точности использовать норму вектора

$$\|x\|_\infty = \max_i(|x_i|). \quad (8.5)$$

4. Повторить пункт 3 задания для плохо обусловленных матриц (см. подразд. 3.6), имеющих порядок от 4 до 40 с шагом 4.

5. Вычислить матрицу  $A^{-1}$  следующими двумя способами.

Способ 1 – через решение системы  $AX = I$ , где  $I$  – единичная матрица.

Способ 2 – через разложение матрицы  $A$  в произведение элементарных матриц, обращение которых осуществляется аналитически, а их произведение даёт матрицу  $A^{-1}$ .

Сравнить затраты машинного времени и точность обращения матриц при использовании указанных способов 1 и 2. Эксперименты провести для случайных матриц порядков от 5 до 100 через 5. Для оценки точности в обоих способах использовать оценочную формулу

$$\|A_{\text{т}}^{-1} - A_{\text{пр}}^{-1}\| \leq \|I - AA_{\text{пр}}^{-1}\| \cdot \|A\|^{-1}. \quad (8.6)$$

Норму матрицы следует вычислять в соответствии со следующим определением:

$$\|A\|_\infty = \max_i \left( \sum_{j=1}^n |a_{ij}| \right), \quad (8.7)$$

где  $A_{\text{т}}^{-1}$  – точное значение обратной матрицы, а  $A_{\text{пр}}^{-1}$  – приближенное значение, полученное в результате обращения каждым из способов 1 и 2.

6. Провести подсчёт фактического числа выполняемых операций умножения и деления при обращении матриц первым и вторым способами, сравнить его с оценочным числом ( $n^3$ ).

**Замечание 8.1.** По ходу проведения численных экспериментов на экран дисплея должны выводиться следующие таблицы.

#### Решение систем линейных алгебраических уравнений:

Порядок	Время	Точность	Теоретическое число операций	Реальное число операций
---------	-------	----------	---------------------------------	----------------------------

Аналогичная таблица должна быть построена для плохо обусловленных матриц.

#### Обращение матриц:

Порядок	Время		Точность		Число операций		
	спос. 1	спос. 2	спос. 1	спос. 2	спос. 1	спос. 2	теорет.

**Замечание 8.2.** Результаты экспериментов необходимо вывести на экран в форме следующих графиков. Для случая обращения матриц при построении графиков использовать данные из второй таблицы.

#### Графики решения систем линейных алгебраических уравнений:

- зависимость реального и оценочного числа операций от порядка матрицы (для разных графиков использовать разные цвета);
- зависимость времени решения от порядка матриц;
- зависимость точности решения от порядка матриц. При построении графиков использовать данные из первой таблицы. Для этого их необходимо записать в текстовый файл.

#### Графики для обращения матриц:

- зависимость реального и оценочного числа операций от порядка матрицы (для разных графиков использовать разные цвета);

- зависимость времени обращения первым и вторым способом от порядка матриц;
- зависимость точности обращения первым и вторым способом от порядка матриц.

## 8.5 Варианты задания на лабораторный проект № 5

В табл. 8.3 приведены 16 номеров вариантов задания на лабораторную работу (проект) № 5.

Если нет других указаний преподавателя, выбирайте ваш вариант по вашему номеру в журнале студенческой группы.

Таблица 8.3. Варианты задания на лабораторный проект № 5

Вид разложения	Окаймление $\alpha$		Окаймление $\beta$	
	Алгоритм <sup>a</sup>	Алгоритм <sup>b</sup>	Алгоритм <sup>c</sup>	Алгоритм <sup>b</sup>
$A = \overline{L}U$	1	2	3	4
$A = L\overline{U}$	5	6	7	8
$A = \overline{U}L$	9	10	11	12
$A = U\overline{L}$	13	14	15	16

$\alpha$  – известной части разложения;

$\beta$  – неизвестной части разложения;

<sup>a</sup> – столбцовый;

<sup>b</sup> – скалярных произведений;

<sup>c</sup> – линейных комбинаций.

## 8.6 Тестовые задачи для проекта № 5

Используйте приводимые ниже задачи в двух режимах:

- для контроля собственного понимания алгоритма,
- для контроля правильности вашего программирования.

**Задача 1**

Для матрицы

$$A = \begin{pmatrix} 2 & 0 & 2 \\ 4 & -1 & 3 \\ -2 & -3 & -2 \end{pmatrix}$$

выполнить следующее:

- Построить  $\overline{L}U$ -разложение матрицы  $A$  ( $\overline{L}$  с единицами на диагонали).
- С помощью  $\overline{L}U$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, 0, -3)^T$ .

- С помощью  $\overline{L}U$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 2**

Для матрицы

$$A = \begin{pmatrix} 3 & 6 & 2 \\ -5 & -10 & -4 \\ 1 & 3 & 1 \end{pmatrix}$$

выполнить следующее:

- Построить  $\overline{U}L$ -разложение матрицы  $A$  ( $\overline{U}$  с единицами на диагонали).
- С помощью  $\overline{U}L$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (10, -16, 5)^T$ .

- С помощью  $\overline{U}L$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 3**

Для матрицы

$$A = \begin{pmatrix} 3 & 0 & 3 \\ -1 & 1 & -2 \\ 1 & 2 & 1 \end{pmatrix}$$

выполнить следующее:

а. Построить  $L\bar{U}$ -разложение матрицы  $A$  ( $\bar{U}$  с единицами на диагонали).

б. С помощью  $L\bar{U}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (0, -2, -2)^T$ .

в. С помощью  $L\bar{U}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

**Задача 4**

Для матрицы

$$A = \begin{pmatrix} -3 & 1 & 1 \\ 2 & 1 & 2 \\ 4 & 0 & 2 \end{pmatrix}$$

выполнить следующее:

а. Построить  $U\bar{L}$ -разложение матрицы  $A$  ( $\bar{L}$  с единицами на диагонали).

б. С помощью  $U\bar{L}$ -разложения матрицы  $A$  решить систему уравнений

$$Ax = b,$$

где вектор  $b = (5, 2, 0)^T$ .

в. С помощью  $U\bar{L}$ -разложения найти матрицу  $A^{-1}$  и вычислить число  $M_A$  обусловленности матрицы  $A$  в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

## 8.7 Заключение по разделу 8

В данном разделе рассмотрены стандартные алгоритмы  $LU$ -разложения матрицы, численно эквивалентные методу Гаусса последовательного исключения неизвестных в системе линейных алгебраических уравнений. Даны теоретические сведения, достаточные для выполнения лабораторного проекта № 5.

Проект № 5 отличается от проекта № 1 или проекта № 4 тем, что в нём предлагаются выполнить программирование  $LU$ -разложения по алгоритмам, относящимся к иной категории, – к так называемым алгоритмам «окаймления».

Идея «окаймления» известна уже более полувека, но тогда она применялась в одном варианте, который можно назвать «окаймление известной части  $LU$ -разложения». В последнее время к этой идеи вернулись, и она получила своё завершающее развитие, например, в форме алгоритма Донгарры-Айзенштадта, так что стало возможным говорить также об окаймлении и неизвестной части  $LU$ -разложения.

В проекте № 5 на этой основе решаются две основные задачи вычислительной линейной алгебры: (1) найти решение СЛАУ и (2) найти обратную матрицу. При этом матрицу системы предлагается формировать тремя различными способами: вводить «вручную» с клавиатуры компьютера, генерировать случайным образом или же из списка специальных – плохо обусловленных – матриц.

Этот проект мы рассматриваем как (дополнительный) проект повышенной сложности (по сравнению с проектом № 1) для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

# 9

## Проект № 6 «Итерационные методы решения систем»

### 9.1 Итерационные методы

Любой итерационный метод (ИМ) есть метод последовательных приближений. Он изначально предполагает наличие методической погрешности (ошибки метода) и этим отличается от прямых методов (ПМ), рассмотренных выше, у которых методическая погрешность равна нулю. Сходящийся ИМ, — а только такие ИМ и представляют интерес, — обеспечивает стремление этой погрешности к нулю по мере роста числа итераций,  $n \rightarrow \infty$ .

ИМ позволяют находить решение с некоторой заранее задаваемой погрешностью и при специальных условиях делают это быстрее, чем ПМ. Они запрашивают меньше ресурсов компьютера при решении больших систем. Алгоритмически ИМ более просты, чем ПМ, и в меньшей степени используют структуру матрицы.

Главные вопросы для любого ИМ: сходится ли он? и какие условия для этого нужны? Однако для ИМ важно знать не только теоретический факт и условия сходимости. Следующие вопросы: какова скорость сходимости? как можно её увеличить? Сходимость простых итерационных методов крайне медленная, особенно, в случае плохо обусловленной матрицы, поэтому эти вопросы крайне важны.

Есть два основных «регулятора» скорости ИМ: так называемая *лидирующая матрица*  $B$  и *скалярный параметр*  $\tau$ . Если  $B \neq I$ , метод называют *неявным ИМ*, поскольку решение на новой итерации приходится искать тем или иным прямым методом применительно к системе с этой матрицей  $B$ , обязанной быть более простой, чем исходная матрица  $A$ . В силу этого неявные методы алгоритмически более сложны, чем *явные ИМ* (в которых  $B = I$ ); однако их преимуще-

ством является существенно более быстрая сходимость. Скалярный параметр  $\tau$ , как и лидирующую матрицу  $B$ , можно выбирать на каждой итерации, т. е. поставить их в зависимость от текущего номера итерации  $n$ . В этом случае *стационарный ИМ* становится *нестационарным итерационным методом*.

Кроме методических погрешностей на результат итерационного процесса влияют *трансформированные погрешности алгоритма* и *погрешности округления*. Положительным качеством ИМ является то, что при их использовании погрешности округления не накапливаются. Для запуска любого ИМ необходимо задавать начальное приближение к искомому решению.

## 9.2 Итерационная формула

ИМ могут применяться как для линейных задач, так и для нелинейных. Основой построения любого ИМ является так называемая *итерационная формула* (ИФ), т. е. формула вида  $x = \varphi(x)$ . Если она есть, то ИМ записывают в виде  $x_{n+1} = \varphi(x_n)$ , где  $n = 0, 1, \dots$ .

Построим ИФ для линейной системы

$$Ax = f$$

с обратимой  $(m \times m)$ -матрицей  $A = [a_{ij}]$ ,  $i, j = 1, 2, \dots, m$ , с неизвестным вектором  $x = (x_1, x_2, \dots, x_m)^T$  и с заданной правой частью  $f = (f_1, f_2, \dots, f_m)^T$ . Пусть  $\forall i \in \{i, j = 1, 2, \dots, m\} \mid a_{ii} \neq 0$ .

Преобразуем данную систему к виду

$$x_i = - \sum_{j=2}^{i-1} \frac{a_{ij}}{a_{ii}} x_j - \sum_{j=i+1}^m \frac{a_{ij}}{a_{ii}} x_j + \frac{f_i}{a_{ii}}, \quad i = 1, 2, \dots, m. \quad (9.1)$$

Условимся считать значение суммы равным нулю, если верхний предел суммирования меньше нижнего. Так, уравнение (9.1) при  $i = 1$  имеет вид:

$$x_1 = - \sum_{j=2}^m \frac{a_{1j}}{a_{11}} x_j + \frac{f_1}{a_{11}}.$$

В дальнейшем верхний индекс будет указывать номер итерации, например,

$$x^n = (x_1^n, x_2^n, \dots, x_m^n)^T,$$

где  $x_i^n$  –  $n$ -я итерация  $i$ -й компоненты вектора  $x$ . Число итераций ограничивают *критерием остановки*: либо  $n < n_{\max}$ , либо задают малое  $\varepsilon$  и проверяют условие

$$\max_{1 \leq i \leq m} |x_i^{n+1} - x_i^n| < \varepsilon.$$

### 9.3 Метод Якоби

В итерационном методе Якоби исходят из записи системы в виде (9.1), при чём итерации определяют формулой

$$x_i^{n+1} = - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^n - \sum_{j=i+1}^m \frac{a_{ij}}{a_{ii}} x_j^n + \frac{f_i}{a_{ii}}, \quad (9.2)$$

$$n = 0, 1, \dots, n_{\max}, \quad i = 1, 2, \dots, m,$$

где начальные значения  $x_i^0, i = 1, 2, \dots, m$  заданы.

### 9.4 Метод Зейделя

Итерационный метод Зейделя определён формулой

$$x_i^{n+1} = - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{n+1} - \sum_{j=i+1}^m \frac{a_{ij}}{a_{ii}} x_j^n + \frac{f_i}{a_{ii}}, \quad (9.3)$$

$$n = 0, 1, \dots, n_{\max}, \quad i = 1, 2, \dots, m,$$

где начальные значения  $x_i^0, i = 1, 2, \dots, m$  заданы.

Для наглядности запишем первые два уравнения системы (9.3):

$$x_1^{n+1} = - \sum_{j=2}^m \frac{a_{1j}}{a_{11}} x_j^n + \frac{f_1}{a_{11}}, \quad (9.4)$$

$$x_2^{n+1} = - \frac{a_{21}}{a_{22}} x_1^{n+1} - \sum_{j=2}^m \frac{a_{2j}}{a_{22}} x_j^n + \frac{f_2}{a_{22}}. \quad (9.5)$$

Первую компоненту  $x_1^{n+1}$  вектора  $x^{n+1}$  находят из (9.4). Для этого используют вектор  $x^n$  и значение  $f_1$ . Для вычисления  $x_2^{n+1}$  по выражению (9.5) используют только что найденное значение  $x_1^{n+1}$  и известные значения  $x_j^n, j = 3, \dots, m$  от предыдущей итерации. Таким образом, компоненты  $x_i^{n+1}$  вектора  $x^{n+1}$  находят из уравнения (9.3) последовательно, начиная с  $i = 1$ .

### 9.5 Матричная запись методов Якоби и Зейделя

Сопоставительный анализ итерационных методов упрощается, если записать их не в координатной, а в матричной форме. Представим матрицу  $A$  в виде суммы трёх матриц

$$A = A_1 + D + A_2, \quad (9.6)$$

где  $D = \text{diag} [a_{11}, a_{22}, \dots, a_{mm}]$  – диагональная матрица с той же главной диагональю, что и матрица  $A$ , матрица  $A_1$  – нижняя треугольная и матрица  $A_2$  – верхняя треугольная, обе с нулевыми главными диагоналями. Используя «расщепление» (9.6), перепишем систему  $Ax = f$  в виде

$$x = -D^{-1}A_1x - D^{-1}A_2x + D^{-1}f.$$

Отсюда видно, что метод Якоби (9.2) представлен формулой

$$x^{n+1} = -D^{-1}A_1x^n - D^{-1}A_2x^n + D^{-1}f,$$

или, что то же самое, формулой

$$Dx^{n+1} + (A_1 + A_2)x^n = f, \quad (9.7)$$

а метод Зейделя (9.3) – формулой

$$x^{n+1} = -D^{-1}A_1x^{n+1} - D^{-1}A_2x^n + D^{-1}f,$$

или, что то же самое, формулой

$$(D + A_1)x^{n+1} + A_2x^n = f. \quad (9.8)$$

Учитывая (9.6), методы (9.7) и (9.8) запишем, соответственно, в виде

$$D(x^{n+1} - x^n) + Ax^n = f, \quad (9.9)$$

$$(D + A_1)(x^{n+1} - x^n) + Ax^n = f. \quad (9.10)$$

Эта запись показывает, что если итерационный метод сходится, то он сходится к решению исходной системы уравнений.

В методы (9.9) и (9.10) можно ввести итерационный параметр  $\tau_{n+1}$  следующим образом:

$$\begin{aligned} & D \frac{x^{n+1} - x^n}{\tau_{n+1}} + Ax^n = f, \\ & (D + A_1) \frac{x^{n+1} - x^n}{\tau_{n+1}} + Ax^n = f. \end{aligned}$$

Приведённые выше методы Якоби и Зейделя относятся к *одношаговым итерационным методам*. В таких методах для нахождения  $x^{n+1}$  используют одну предыдущую итерацию  $x^n$ . *Многошаговые итерационные методы* определяют  $x^{n+1}$  через значения  $x^k$  на двух и более предыдущих итерациях, так что  $l$ -шаговый ИМ выглядит как некоторая зависимость  $x^{n+1} = \varphi(x^n, \dots, x^{n-l+1})$ .

## 9.6 Каноническая форма одношаговых ИМ

*Канонической формой одношагового итерационного метода для решения системы  $Ax = f$  называют его представление в виде*

$$B_{n+1} \frac{x^{n+1} - x^n}{\tau_{n+1}} + Ax^n = f, \quad n = 0, 1, \dots, n_{\max}. \quad (9.11)$$

Здесь  $B_{n+1}$  – *лидирующая матрица*, задающая тот или иной итерационный метод,  $\tau_{n+1}$  – *итерационный параметр*. Предполагается, что дано начальное приближение  $x^0$  и что существуют  $B_{n+1}$  и  $\tau_{n+1}$ ,  $n = 0, 1, \dots, n_{\max}$ . Тогда из уравнения (9.11) можно последовательно определить все  $x^{n+1}$ ,  $n = 0, 1, \dots, n_{\max}$ .

Для нахождения  $x^{n+1}$  по известным  $f$  и  $x^n$  достаточно решить систему

$$B_{n+1}x^{n+1} = F_n$$

с правой частью  $F_n = (B_{n+1} - \tau_{n+1}A)x^n + \tau_{n+1}f$ .

*Стационарный ИМ* определяется выполнением двух условий:  $B_{n+1} = B = \text{const}$  и  $\tau_{n+1} = \tau = \text{const}$ ; иначе имеем *нестационарный ИМ*.

## 9.7 Методы простой итерации, Ричардсона и Юнга

*Методом простой итерации* называют явный метод

$$\frac{x^{n+1} - x^n}{\tau} + Ax^n = f \quad (9.12)$$

с постоянным параметром  $\tau$ . Явный метод

$$\frac{x^{n+1} - x^n}{\tau_{n+1}} + Ax^n = f \quad (9.13)$$

с переменным  $\tau_{n+1}$  есть *итерационный метод Ричардсона*. Юнг усовершенствовал метод Зейделя, введя в него параметр  $\omega > 0$ . Этот метод получил название *метод верхней релаксации*:

$$(D + \omega A_1) \frac{x^{n+1} - x^n}{\omega} + Ax^n = f. \quad (9.14)$$

Расчётная схема для (9.14) с учётом (9.6) получается в виде

$$(I + \omega D^{-1}A_1)x^{n+1} = ((1 - \omega)I - \omega D^{-1}A_2)x^n + \omega D^{-1}f.$$

В покомпонентной записи имеем

$$x_i^{n+1} + \omega \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{n+1} = (1 - \omega)x_i^n - \omega \sum_{j=i+1}^m \frac{a_{ij}}{a_{ii}} x_j^n + \omega \frac{f_i}{a_{ii}}, \quad i = 1, 2, \dots, m.$$

Последовательно, начиная с  $i = 1$ , находим все  $x_i^{n+1}$ , например, первые два:

$$\begin{aligned} x_1^{n+1} &= (1 - \omega)x_1^n - \omega \sum_{j=2}^m \frac{a_{1j}}{a_{11}} x_j^n + \omega \frac{f_1}{a_{11}}, \\ x_2^{n+1} &= -\omega \frac{a_{21}}{a_{11}} x_1^{n+1} + (1 - \omega)x_2^n - \omega \sum_{j=3}^m \frac{a_{2j}}{a_{22}} x_j^n + \omega \frac{f_2}{a_{22}}. \end{aligned}$$

## 9.8 Сходимость итерационных методов

Запишем стационарный одношаговый итерационный метод (9.11) в терминах погрешности  $z^n = x^n - x$ :

$$B \frac{z^{n+1} - z^n}{\tau} + Az^n = 0, \quad n = 0, 1, \dots, \quad z^0 = x^0 - x. \quad (9.15)$$

**Теорема 9.1** ([5]). Пусть  $A = A^T > 0$ ,  $\tau > 0$  и выполнено неравенство

$$B - 0.5\tau A > 0. \quad (9.16)$$

Тогда итерационный метод (9.11) сходится.

**Следствие 9.1.** Пусть  $A = A^T > 0$ . Тогда метод Якоби сходится [5], если  $A$  – матрица с диагональным преобладанием, т. е. при условии

$$a_{ii} > \sum_{j \neq i} |a_{ij}|, \quad i = 1, 2, \dots, m. \quad (9.17)$$

**Следствие 9.2.** Пусть  $A = A^T > 0$ . Тогда метод верхней релаксации

$$(D + \omega A_1) \frac{x^{n+1} - x^n}{\omega} + Ax^n = f$$

сходится при условии  $0 < \omega < 2$ . В частности, метод Зейделя сходится [5].

**Следствие 9.3.** Пусть  $A = A^T > 0$ . Тогда для метода простой итерации

$$\frac{x^{n+1} - x^n}{\tau} + Ax^n = f$$

необходимым и достаточным условием сходимости является неравенство

$$0 < \tau < 2/\lambda_{\max},$$

где  $\lambda_{\max}$  – наибольшее по абсолютному значению собственное число матрицы  $A$ , называемое также *спектральным радиусом*  $\rho(A)$  матрицы  $A$  [5].

*Сходимость итерационного метода* (9.11) означает, что  $z^n \rightarrow 0$  в некоторой норме при  $n \rightarrow \infty$ . Переписав уравнение (9.15), получим:

$$z^{n+1} = Sz^n, \quad n = 0, 1, \dots, \quad (9.18)$$

где

$$S = I - \tau B^{-1}A \quad (9.19)$$

называют *переходной матрицей погрешности* от  $n$ -й итерации к  $(n + 1)$ -й.

**Теорема 9.2** ([5]). Итерационный метод

$$B \frac{x^{n+1} - x^n}{\tau} + Ax^n = f, \quad n = 0, 1, \dots, \quad (9.20)$$

сходится при любом начальном приближении тогда и только тогда, когда все собственные значения матрицы (9.19) по модулю меньше единицы.

## 9.9 Скорость сходимости итерационных методов

Теорема 9.2 о сходимости имеет принципиальное значение и накладывает минимальные ограничения на матрицы  $A$  и  $B$ . Однако её непосредственное применение к конкретным итерационным методам не всегда возможно, так как исследование спектра матрицы (9.19) является более трудоемкой задачей, чем решение исходной системы  $Ax = f$ .

Будем рассматривать решение  $x$  системы и последовательные приближения  $x^n$  как элементы евклидова пространства, а матрицы  $A$ ,  $B$  и другие – как операторы, действующие в нём.

**Замечание 9.1.** Для двух симметрических матриц  $A$  и  $B$  неравенство  $A \geq B$  означает, что  $(Ax, x) \geq (Bx, x)$  для всех  $x \in \mathbb{E}$ . В случае некоторой симметрической положительно определённой матрицы  $D$  будем пользоваться обобщённой нормой вектора  $\|y\|_D = \sqrt{(Dy, y)}$ .

**Теорема 9.3** ([5]). Пусть  $A$  и  $B$  – симметрические положительно определённые матрицы, для которых справедливы неравенства

$$\gamma_1 B \leq A \leq \gamma_2 B, \quad (9.21)$$

где  $\gamma_1, \gamma_2$  — положительные константы и  $\gamma_1 < \gamma_2$ . При  $\tau = 2/(\gamma_1 + \gamma_2)$  итерационный метод (9.20) сходится, и для погрешности справедливы оценки:

$$\|x^n - x\|_A \leq \rho^n \|x^0 - x\|_A, \quad n = 0, 1, \dots,$$

$$\|x^n - x\|_B \leq \rho^n \|x^0 - x\|_B, \quad n = 0, 1, \dots,$$

где

$$\rho = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\gamma_1}{\gamma_2}. \quad (9.22)$$

Пусть

$$A\mu = \lambda B\mu. \quad (9.23)$$

Тогда

$$\gamma_1(B\mu, \mu) \leq (A\mu, \mu) = \lambda(B\mu, \mu) \leq \gamma_2(B\mu, \mu)$$

и

$$\gamma_1 \leq \lambda_{\min}(B^{-1}A), \quad \gamma_2 \geq \lambda_{\max}(B^{-1}A), \quad (9.24)$$

где  $\lambda_{\min}(B^{-1}A)$  и  $\lambda_{\max}(B^{-1}A)$  — минимальное и максимальное по абсолютному значению собственные числа в обобщённой задаче (9.23) на собственные значения.

Таким образом, наиболее точными константами, с которыми выполняются неравенства (9.21), являются константы

$$\gamma_1 = \lambda_{\min}(B^{-1}A), \quad \gamma_2 = \lambda_{\max}(B^{-1}A).$$

В этом случае параметр

$$\tau = \frac{2}{\lambda_{\min}(B^{-1}A) + \lambda_{\max}(B^{-1}A)} \quad (9.25)$$

называется оптимальным итерационным параметром, минимизирующим

$$\rho = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\gamma_1}{\gamma_2}$$

на множестве всех положительных  $\gamma_1, \gamma_2$ , удовлетворяющих условиям (9.24).

В случае метода простой итерации ( $B = I$ ) получаем два следствия.

**Следствие 9.4.** Если  $A^T = A > 0$ , то для метода простой итерации

$$\frac{x^{n+1} - x^n}{\tau} + Ax^n = f$$

при

$$\tau = \tau_0 = \frac{2}{\lambda_{\min}(A) + \lambda_{\max}(A)}$$

справедлива оценка

$$\|x^n - x\| \leq \rho_0^n \|x^0 - x\|,$$

где [5]

$$\rho_0 = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\lambda_{\min}(A)}{\lambda_{\max}(A)}.$$

**Следствие 9.5.** Для симметрической матрицы  $A$  и  $\tau_0 = 2/(\lambda_{\min}(A) + \lambda_{\max}(A))$  справедливо равенство

$$\|I - \tau_0 A\| = \rho_0,$$

где [5]

$$\rho_0 = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\lambda_{\min}(A)}{\lambda_{\max}(A)}.$$

В приложениях часто встречаются задачи с *плохо обусловленной матрицей*  $A$ , когда соотношение  $\lambda_{\max}(A)/\lambda_{\min}(A)$  велико. В этом случае число  $\rho_0$  близко к единице, и метод простой итерации сходится медленно. Число итераций  $n_0(\varepsilon)$ , которое требуется в случае малых  $\xi$  для достижения заданной точности  $\varepsilon$ , т. е. для достижения оценки

$$\|x^n - x\| \leq \varepsilon \|x^0 - x\|,$$

получается из условия  $\rho_0^n < \varepsilon$  в виде  $n \geq n_0(\varepsilon)$ , где

$$n_0(\varepsilon) = \frac{\ln(1/\varepsilon)}{\ln(1/\rho_0)}.$$

Отсюда при малых  $\xi$  имеем

$$n_0(\varepsilon) \approx \frac{\ln(1/\varepsilon)}{2\xi} = O\left(\frac{1}{\xi}\right).$$

Это свидетельствует о том, что метод простой итерации в случае малых  $\xi$  является медленно сходящимся методом. Ускорить сходимость можно двумя способами: применяя неявный итерационный метод и/или делая  $\tau = \tau_{n+1}$  зависящим от номера итерации.

## 9.10 Итерационные методы вариационного типа

Найти минимальное и максимальное по абсолютному значению собственные числа в обобщённой задаче (9.23) на собственные значения бывает сложно, а без них невозможно задать наилучшее значение итерационного параметра (9.25). В таких случаях можно использовать другой класс итерационных методов — *методы вариационного типа*. Здесь на каждой итерации

$$B \frac{x^{k+1} - x^k}{\tau_{k+1}} + Ax^k = f, \quad (9.26)$$

для параметра  $\tau_{k+1}$  выбирают то значение, которое минимизирует предопределенный критерий качества, связанный с погрешностью  $\|x^{k+1} - x\|_D$ , при условии, что предыдущая итерация уже состоялась с погрешностью  $\|x^k - x\|_D$ . В зависимости от выбора матриц  $D$  и  $B$  получают различные методы этого типа.

### Метод минимальных невязок

Рассмотрим уравнение  $Ax = f$  с  $A = A^T > 0$ . Разность

$$r_k = Ax^k - f, \quad (9.27)$$

которая получается при подстановке приближённого значения  $x^k$  в это уравнение, называют *невязкой*. Погрешность  $z_k = x^k - x$  и невязка  $r_k$  связаны равенством  $Az_k = r_k$ . Представим явный итерационный метод

$$\frac{x^{k+1} - x^k}{\tau_{k+1}} + Ax^k = f \quad (9.28)$$

в виде

$$x^{k+1} = x^k - \tau_{k+1}r_k. \quad (9.29)$$

*Метод минимальных невязок* есть метод (9.28), в котором параметр  $\tau_{k+1}$  минимизирует  $\|r_{k+1}\|$  при заданной норме  $\|r_k\|$  невязки текущего шага. Найдём это значение. Из (9.29) получаем:

$$\begin{aligned} Ax^{k+1} &= Ax^k - \tau_{k+1}Ar_k, \\ r_{k+1} &= r_k - \tau_{k+1}Ar_k. \end{aligned} \quad (9.30)$$

Возводя обе части уравнения (9.30) скалярно в квадрат, получим

$$\|r_{k+1}\|^2 = \|r_k\|^2 - 2\tau_{k+1}(r_k, Ar_k) + \tau_{k+1}^2\|Ar_k\|^2.$$

Отсюда видно, что  $\|r_{k+1}\|$  достигает минимума при

$$\tau_{k+1} = \frac{(Ar_k, r_k)}{\|Ar_k\|^2}. \quad (9.31)$$

Таким образом, в методе минимальных невязок переход от  $k$ -й итерации к  $(k + 1)$ -й осуществляется по следующему алгоритму:

- по найденному значению  $x^k$  вычисляют вектор невязки  $r_k = Ax^k - f$ ,
- по формуле (9.31) находят параметр  $\tau_{k+1}$ ,
- по формуле (9.29) определяют вектор  $x^{k+1}$ .

**Теорема 9.4** ([5]). Пусть  $A$  – симметрическая положительно определённая матрица. Для погрешности метода минимальных невязок справедлива оценка

$$\|A(x^n - x)\| \leq \rho_0^n \|A(x^0 - x)\|, \quad n = 0, 1, \dots,$$

где

$$\rho = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\lambda_{\min}(A)}{\lambda_{\max}(A)}.$$

Иными словами, метод минимальных невязок сходится с той же скоростью, что и метод простой итерации с оптимальным параметром  $\tau$ .

## Метод минимальных поправок

Запишем неявный итерационный метод (9.26) в виде

$$x^{k+1} = x^k - \tau B^{-1} r_k,$$

где  $r_k = Ax^k - f$  – невязка. Вектор  $\omega_k = B^{-1}r_k$  называют *поправкой итерационного метода* на  $(k + 1)$ -й итерации. Поправка  $\omega_k$  удовлетворяет тому же уравнению, что и погрешность  $z_k = x^k - x$  неявного метода, т. е. уравнению

$$B \frac{\omega_{k+1} - \omega_k}{\tau_{k+1}} + A\omega_k = 0. \quad (9.32)$$

Пусть  $B$  – симметрическая положительно определённая матрица. Тогда *метод минимальных поправок* – это метод (9.26), в котором параметр  $\tau_{k+1}$  минимизирует норму  $\|\omega_{k+1}\|_B = (B\omega_{k+1}, \omega_{k+1})^{1/2}$  при ранее полученном векторе  $\omega^k$ . В случае  $B = I$  метод минимальных поправок совпадает с методом минимальных невязок.

Перепишем (9.32) в виде

$$\omega_{k+1} = \omega_k - \tau_{k+1} B^{-1} A \omega_k$$

и вычислим

$$\|\omega_{k+1}\|_B^2 = \|\omega_k\|_B^2 - 2\tau_{k+1}(A\omega_k, \omega_k) + \tau_{k+1}^2(B^{-1}A\omega_k, A\omega_k).$$

Минимум  $\|\omega_{k+1}\|_B^2$  достигается, если и только если

$$\tau_{k+1} = \frac{(A\omega_k, \omega_k)}{(B^{-1}A\omega_k, A\omega_k)}. \quad (9.33)$$

Для реализации метода минимальных поправок требуется на каждой итерации решать систему уравнений  $B\omega_k = r_k$  относительно поправки  $\omega_k$  и затем решать систему уравнений  $Bv_k = A\omega_k$ , откуда находят вектор  $v_k = B^{-1}A\omega_k$ , необходимый для вычисления параметра  $\tau_{k+1}$ .

**Теорема 9.5 ([5]).** Пусть  $A$  и  $B$  – симметрические положительно определенные матрицы и  $\lambda_{\min}(B^{-1}A)$ ,  $\lambda_{\max}(B^{-1}A)$  – наименьшее и наибольшее собственные значения в задаче  $Ax = \lambda Bx$ . Для погрешности метода минимальных поправок справедлива оценка

$$\|A(x^n - x)\|_{B^{-1}} \leq \rho_0^n \|A(x^0 - x)\|_{B^{-1}}, \quad n = 0, 1, \dots,$$

где

$$\rho_0 = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\lambda_{\min}(B^{-1}A)}{\lambda_{\max}(B^{-1}A)}.$$

### Метод скорейшего спуска

Возьмём явный метод (9.13) и выберем итерационный параметр  $\tau_{k+1}$  из условия минимума  $\|z_{k+1}\|_A$  при заданном векторе  $z_k$ , где  $z_{k+1} = x^{k+1} - x$ . Поскольку погрешность  $z_k$  удовлетворяет уравнению

$$z_{k+1} = z_k - \tau_{k+1}Az_k,$$

имеем

$$\|z_{k+1}\|_A^2 = \|z_k\|_A^2 - 2\tau_{k+1}(Az_k, Az_k) + \tau_{k+1}^2(A^2z_k, Az_k).$$

Минимум нормы  $\|z_{k+1}\|_A^2$  достигается при  $\tau_{k+1} = \frac{(Az_k, Az_k)}{(A^2z_k, Az_k)}$ .

Величина  $z_k = x^k - x$  неизвестна, но  $Az_k = r_k = Ax^k - f$ . Поэтому вычисление  $\tau_{k+1}$  проводят по формуле

$$\tau_{k+1} = \frac{(r_k, r_k)}{(Ar_k, r_k)}.$$

**Теорема 9.6 ([5]).** Для погрешности явного метода скорейшего спуска справедлива оценка

$$\|x^n - x\|_A \leq \rho_0^n \|x^0 - x\|_A, \quad n = 0, 1, \dots,$$

где

$$\rho_0 = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\lambda_{\min}(A)}{\lambda_{\max}(A)}.$$

Если вместо (9.13) взять неявный метод (9.26) и параметр  $\tau_{k+1}$  выбирать из условия минимума  $\|z_{k+1}\|_A$ , то получим *неявный метод наискорейшего спуска*. Для него

$$\|z_{k+1}\|_A^2 = \|z_k\|_A^2 - 2\tau_{k+1}(Az_k, B^{-1}Az_k) + \tau_{k+1}^2(AB^{-1}Az_k, B^{-1}Az_k),$$

или

$$\|z_{k+1}\|_A^2 = \|z_k\|_A^2 - 2\tau_{k+1}(r_k, \omega_k) + \tau_{k+1}^2(A\omega_k, \omega_k).$$

Следовательно, норма  $\|z_{k+1}\|_A^2$  будет минимальной при

$$\tau_{k+1} = \frac{(r_k, \omega_k)}{(A\omega_k, \omega_k)}.$$

**Теорема 9.7 ([5]).** Для неявного метода скорейшего спуска справедлива оценка

$$\|x^n - x\|_A \leq \rho_0^n \|x^0 - x\|_A, \quad n = 0, 1, \dots,$$

где

$$\rho_0 = \frac{1 - \xi}{1 + \xi}, \quad \xi = \frac{\lambda_{\min}(B^{-1}A)}{\lambda_{\max}(B^{-1}A)}.$$

## Метод сопряжённых градиентов

Этот метод исходит из задачи минимизации функции

$$J(x) = \frac{1}{2}(Ax, x) - (b, x), \tag{9.34}$$

решение которой совпадает с решением системы

$$Ax = f, \quad A = A^T > 0. \tag{9.35}$$

Полный вывод метода сопряжённых градиентов можно найти в [5]. Опуская детали, приведём окончательный результат.

Метод сопряжённых градиентов для решения системы  $Ax = f$  состоит в вычислениях по следующим формулам:

$$\left. \begin{array}{l} r^k = b - Ax^k, \quad k = 0, 1, \dots, \\ p^{k+1} = r^k + \beta_{k+1} p^k, \quad k = 1, 2, \dots, \quad p^1 = r^0, \\ x^{k+1} = x^k + \alpha_{k+1} p^{k+1}, \quad k = 0, 1, \dots, \quad x^0 = 0, \\ \alpha_{k+1} = (r^k, p^{k+1}) / (p^{k+1}, Ap^{k+1}), \quad k = 0, 1, \dots, \\ \beta_{k+1} = -(Ap^k, r^k) / (Ap^k, p^k), \quad k = 1, 2, \dots. \end{array} \right\} \quad (9.36)$$

**Теорема 9.8 ([5]).** Для метода сопряжённых градиентов (9.36) справедливо

$$\|x^k - x\|_A \leq 2 \left[ (1 - \sqrt{\lambda_{\min}/\lambda_{\max}}) / (1 + \sqrt{\lambda_{\min}/\lambda_{\max}}) \right]^k \|x\|_A,$$

где  $\lambda_{\min}$  и  $\lambda_{\max}$  – минимальное и максимальное собственные значения матрицы  $A$ .

Следуя [5], преобразуем соотношения (9.36). В этих соотношениях наиболее трудоёмкими являются две операции: вычисление векторов  $Ax^k$  и  $Ap^k$ . Однако операцию вычисления вектора  $Ax^k$  можно исключить. Поскольку этот вектор нужен только для вычислении невязки  $r^k$ , то можно заменить первую из формул (9.36) на

$$r^k = r^{k-1} - \alpha_k Ap^k, \quad k = 1, 2, \dots, \quad r^0 = b. \quad (9.37)$$

Преобразуем формулы для вычисления параметров  $\alpha_{k+1}$  и  $\beta_{k+1}$ . Подставляя второе из соотношений (9.36) в четвёртое, найдём

$$\alpha_{k+1} = (r^k, r^k) / (p^{k+1}, Ap^{k+1}), \quad k = 0, 1, \dots. \quad (9.38)$$

Заменяя здесь  $k + 1$  на  $k$  и подставляя полученное выражение для  $(p^k, Ap^k)$  в последнее из соотношений (9.36), получим

$$\beta_{k+1} = -\alpha_k \frac{(Ap^k, r^k)}{(r^{k-1}, r^{k-1})}.$$

Теперь подставим сюда вместо  $Ap^k$  его выражение из (9.37).

**Теорема 9.9 ([5]).** После  $k$  шагов метода сопряжённых градиентов невязки  $r^0, r^1, \dots, r^k$  взаимно ортогональны.

Принимая это во внимание, найдём

$$\beta_{k+1} = \frac{(r^k, r^k)}{(r^{k-1}, r^{k-1})}, \quad k = 1, 2, \dots. \quad (9.39)$$

С учётом (9.37)–(9.39) формулы метода сопряжённых градиентов (9.36) преобразуются к виду

$$\left. \begin{array}{l} r^k = r^{k-1} - \alpha_k A p^k, \quad k = 1, 2, \dots, \quad r^0 = b, \\ p^{k+1} = r^k + \beta_{k+1} p^k \quad k = 1, 2, \dots, \quad p^1 = r^0, \\ x^{k+1} = x^k + \alpha_{k+1} p^{k+1}, \quad k = 0, 1, \dots, \quad x^0 = 0, \\ \alpha_{k+1} = \|r^k\|^2 / (p^{k+1}, A p^{k+1}), \quad k = 0, 1, \dots, \\ \beta_{k+1} = \|r^k\|^2 / \|r^{k-1}\|^2, \quad k = 1, 2, \dots. \end{array} \right\} \quad (9.40)$$

Легко проверить, что эти вычисления проводят в следующем порядке:

$$\begin{aligned} r^0 &= b, & p^1 &= r^0, & Ap^1, & \alpha_1, & x^1, \\ r^1, & \beta_2, & p^2, & Ap^2, & \alpha_2, & x^2, & \dots \end{aligned}$$

## 9.11 Другие методы

Область итерационных методов решения систем линейных алгебраических уравнений обширна. Она включает гораздо большее количество методов, чем то, что приведено выше.

В итерационных методах нашли применение полиномы Чебышёва, благодаря которым можно решать задачу оптимального выбора итерационных параметров как для явных ИМ, так и для неявных ИМ [5].

Стационарные методы, широко применяющиеся в 1950–1980 годах, сейчас чаще применяются [23] как средство сглаживания в многосеточных алгоритмах [21, 22, 31] или для предобусловливания в алгоритмах Крылова [26].

Идея сопряжённых градиентов [25] оказалась очень плодотворной, и наиболее широкое воплощение она нашла при опоре на метод подпространств Крылова, который является одним из методов решения проблемы собственных значений и собственных векторов для очень больших разреженных матриц [30]. Переход к методу подпространств Крылова в этой проблеме вызван тем, что преобразования подобия, лежащие в основе её решения для небольших матриц, выполнять для очень больших матриц практически невозможно. В то же время достаточно легко выполнять однотипные операции умножения матрицы на вектор: взять вектор  $x$  и затем, умножая слева на  $A$ , построить последовательность Крылова  $x, Ax, A^2x, A^3x, \dots$  и, соответственно, получить пространства Крылова

$$\mathcal{K}_j(A, x) = \text{span} \{x, Ax, A^2x, A^3x, \dots, A^{j-1}x\}.$$

В настоящее время алгоритмы Крылова с предобусловливанием применяются в большинстве итерационных методов решения больших разреженных линейных систем [23]. Успешной альтернативой методам Крылова являются *многосеточные методы*, по которым за последние 30–40 лет появилось огромное число публикаций [21, 22, 23, 31].

Вместе с этими мощными ветвями роста, в практике решения линейных систем итерационными методами встречаются решения, которые могут быть классифицированы как *Inventive Math*. Это решения, по которым пока не найдено строгих доказательств, но которые подтверждают свою работоспособность методом широкого вычислительного эксперимента. Примером такого подхода является метод *делинеаризации* для линейных систем [27, 28, 29].

## 9.12 Задание на лабораторный проект № 6

Написать и отладить программу, реализующую ваш вариант задания в соответствии с табл. 9.1 (см. ниже стр. 369), включающий два итерационных метода для численного решения систем линейных алгебраических уравнений  $Ax = f$  с квадратной матрицей  $A$  и отыскания обратной матрицы  $A^{-1}$ . Предусмотреть сообщение о невозможности решения указанных задач из-за превышения допустимого числа итераций. Отделить основные части программы:

- а) подпрограмму решения систем линейных алгебраических уравнений;
- б) подпрограмму обращения матриц;
- в) сервисные подпрограммы.

Уделить особое внимание эффективности программы (в смысле экономии оперативной памяти и скорости решения указанных выше задач). Предусмотреть пошаговое выполнение алгоритма с выводом  $x^k$  на каждой итерации (для тестовой задачи небольшой размерности, см. ниже п. 4).

В качестве  $\varepsilon$  (см. критерий остановки в подразд. 9.2) для обоих итерационных методов использовать погрешность решения данной системы линейных алгебраических уравнений (СЛАУ) методом исключения переменных из лабораторной работы № 1.

Выполнить следующие пункты задания:

1. Провести подсчёт фактического количества операций умножения и деления, выполняемых при решении системы линейных алгебраических уравнений с выводом результата на экран. Сравнить с методом исключения переменных из лабораторной работы № 1. Вывести таблицу и график.
2. Оценить скорость решения задач, т. е. определить время, затраченное на

решение СЛАУ, и время, затраченное на обращение матриц. Для этого спроектировать и провести эксперимент, который охватывает матрицы порядка от 10 до 200 (через 10 порядков). Представить результаты в виде таблицы и графика зависимости времени выполнения от порядка матриц для трёх алгоритмов (двух итерационных методов, соответствующих варианту, и методу исключения переменных из лабораторной работы № 1). Таблицу и графики вывести на экран.

3. Оценить точность решения систем линейных алгебраических уравнений, указанных в п. 2. Для этого сгенерировать случайные матрицы  $A$ , задать точное решение  $x^*$  и образовать правые части  $f = Ax^*$ . Провести анализ точности вычисленного решения  $x$  от порядка матрицы для трёх алгоритмов (аналогично п. 2). В качестве точного решения взять вектор  $x^* = (1, 2, \dots, m)^T$ , где  $m$  – порядок матрицы. Для оценки точности решения использовать норму вектора

$$\|x\| = \max_i |x_i|.$$

Результаты по п. 3 представить в виде таблицы и графиков.

**Замечание 9.2.** Для проведения вычислительного эксперимента по пп. 2 и 3 применять симметрические положительно определённые матрицы  $A$  с диагональным преобладанием. Для заполнения матрицы  $A$  использовать случайные числа из диапазона от  $-100$  до  $100$ . Сначала заполнить нижнюю треугольную часть матрицы  $A$ , т. е., элементы  $a_{ij}$ , где  $i > j$ . Верхнюю треугольную часть, где  $i < j$ , заполнить симметрично нижней части. Затем заполнить диагональ. В качестве диагонального элемента  $a_{ii}$ ,  $i = 1, 2, \dots, m$ , выбрать случайное число из интервала

$$\left[ \sum_{j \neq i} |a_{ij}| + 1, \sum_{j \neq i} |a_{ij}| + 101 \right],$$

чтобы обеспечить выполнение условия

$$a_{ii} \geq \sum_{j \neq i} |a_{ij}| + 1,$$

гарантирующего положительную определённость матрицы  $A$ .

4. До проведения вычислительного эксперимента по пп. 2 и 3 выполнить отладку программы. Для отладки программы, а также для сравнительного тестирования двух заданных итерационных методов использовать следующую тестовую задачу [20]:

$$A = \begin{bmatrix} 4 & 0 & 1 & 1 \\ 0 & 4 & 0 & 1 \\ 1 & 0 & 4 & 0 \\ 1 & 1 & 0 & 4 \end{bmatrix}, \quad f = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad x^* = \begin{bmatrix} -41/209 = -0.196172 \\ 53/209 = 0.253589 \\ 167/209 = 0.799043 \\ 206/209 = 0.985646 \end{bmatrix}.$$

5. Для тех вариантов задания, в которых присутствует метод Юнга (верхней релаксации), провести специальный вычислительный эксперимент решения тестовой задачи по п. 4. Цель этого эксперимента — исследование скорости сходимости метода в зависимости от коэффициента релаксации  $\omega$  в формуле (9.14).

Изменение параметра  $\omega$  организовать с шагом  $\Delta\omega = 0.05$  равномерно в интервале теоретической сходимости метода:  $0 < \omega < 2$ , т. е., по алгоритму:

$$\begin{aligned} \omega_0 &:= \omega_{\text{start}} \\ \text{Для } i &= 0, 1, \dots, 20 \text{ выполнять} \\ \omega_{i+1} &:= \omega_i + \Delta\omega \end{aligned}$$

Стартовое значение задавать с клавиатуры, например,  $\omega_{\text{start}} = 0.50$ .

6. Повторить п. 3 задания для плохо обусловленных матриц (см. подразд. 3.6 лабораторной работы № 1), имеющих порядок от 4 до 40.

7. Вычислить матрицу  $A^{-1}$  двумя способами:

1) через решение системы  $AX = I$  на основе метода исключения Гаусса из лабораторной работы № 1 (в соответствии со своим вариантом);

2) через решение системы  $AX = I$  на основе любого из двух заданных итерационных методов.

Сравнить затраты машинного времени и точность обращения способами 1) и 2). Эксперименты провести для матриц порядков от 10 до 100 через 10, сгенерированных согласно замечанию 9.2. Для оценки точности в обоих способах воспользоваться формулой из лабораторной работы (проекта) № 1.

## 9.13 Варианты задания на лабораторный проект № 6

По теме «Итерационные методы» студентам предлагается 15 вариантов лабораторной работы № 7, сведённых в табл. 9.1.

Если нет других указаний преподавателя, выбирайте ваш вариант из табл. 9.1 по вашему номеру в журнале студенческой группы.

Таблица 9.1. Варианты задания на лабораторный проект № 7

Варианты итерационных методов	a	b	c	d	e	f	g
a	-	13	14	1	2	3	4
b	-	-	15	5	6	7	8
c	-	-	-	9	10	11	12

<sup>a</sup> – метод Якоби;<sup>b</sup> – метод Зейделя;<sup>c</sup> – метод Юнга;<sup>d</sup> – метод минимальных невязок;<sup>e</sup> – метод минимальных поправок;<sup>f</sup> – метод скорейшего спуска;<sup>g</sup> – метод сопряженных градиентов.

## 9.14 Тестовые задачи для проекта № 6

Используйте приводимые ниже задачи в двух режимах:

- для контроля собственного понимания алгоритма,
- для контроля правильности вашего программирования.

### Задача 1

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 1 & -1 \\ -1 & 5 & 0.5 \\ 1 & 1 & -10 \end{pmatrix}$$

и вектор  $b = (-18, 1, 18)^T$ , выполнить следующее:

- Сформулировать метод Якоби в координатном и каноническом виде.
- Определить, является ли он сходящимся с нулевым начальным приближением, т.е.  $x^0 = (0, 0, 0)^T$ ? Ответ обосновать.

- в. Вычислить две итерации по методу Якоби и найти апостериорную оценку ошибки на каждой из них в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 2

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 5 & -1 & 0 \\ -1 & 4 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

и вектор  $b = (4, 2, -1)^T$ , выполнить следующее:

- а. Сформулировать метод Зейделя в координатном и каноническом виде.
- б. Определить, является ли он сходящимся с нулевым начальным приближением, т. е.  $x^0 = (0, 0, 0)^T$ ? Ответ обосновать.
- в. Вычислить две итерации по методу Зейделя и найти апостериорную оценку ошибки на каждой из них в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 3

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} -10 & 3 & -1 \\ 1 & -5 & 1 \\ 1 & 1 & 10 \end{pmatrix}$$

и вектор  $b = (5, -7, -19)^T$ , выполнить следующее:

- а. Сформулировать метод Якоби в координатном и каноническом виде.
- б. Определить, является ли он сходящимся с нулевым начальным приближением, т. е.  $x^0 = (0, 0, 0)^T$ ? Ответ обосновать.

- в. Вычислить две итерации по методу Якоби и найти апостериорную оценку ошибки на каждой из них в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 4

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 4 & 0 & -1 \\ 0 & 5 & 2 \\ -1 & 2 & 10 \end{pmatrix}$$

и вектор  $b = (-3, -2, -9)^T$ , выполнить следующее:

- а. Сформулировать метод Зейделя в координатном и каноническом виде.
- б. Определить, является ли он сходящимся с нулевым начальным приближением, т.е.  $x^0 = (0, 0, 0)^T$ ? Ответ обосновать.
- в. Вычислить две итерации по методу Зейделя и найти апостериорную оценку ошибки на каждой из них в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 5

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 2 & 0 \\ 2 & 5 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

и вектор  $b = (8, -4, 3)^T$ , выполнить следующее:

- а. Сформулировать метод Зейделя в координатном и каноническом виде.
- б. Определить, является ли он сходящимся с нулевым начальным приближением, т.е.  $x^0 = (0, 0, 0)^T$ ? Ответ обосновать.

- в. Вычислить две итерации по методу Зейделя и найти апостериорную оценку ошибки на каждой из них в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 6

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 1 & -1 \\ -1 & 5 & 0.5 \\ 1 & 1 & 10 \end{pmatrix}$$

и вектор  $b = (-9, 6, 0)^T$ , выполнить следующее:

- а. Сформулировать метод минимальных невязок в каноническом виде.
- б. Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?
- в. Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 7

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 1 & -1 \\ -1 & 5 & 0.5 \\ 1 & 1 & 10 \end{pmatrix}$$

и вектор  $b = (-9, 6, 0)^T$ , выполнить следующее:

- а. Сформулировать явный метод скорейшего спуска в каноническом виде.
- б. Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?

- в. Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 8

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 3 & -1 \\ 1 & 5 & 1 \\ 2 & 1 & 10 \end{pmatrix}$$

и вектор  $b = (11, 0, -8)^T$ , выполнить следующее:

- а. Сформулировать метод минимальных невязок в каноническом виде.
- б. Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?
- в. Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 9

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 3 & -1 \\ 1 & 5 & 1 \\ 2 & 1 & 10 \end{pmatrix}$$

и вектор  $b = (11, 0, -8)^T$ , выполнить следующее:

- а. Сформулировать явный метод скорейшего спуска в каноническом виде.
- б. Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?

- в. Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 10

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 4 & 0 & -1 \\ 0 & 5 & 2 \\ -1 & 2 & 10 \end{pmatrix}$$

и вектор  $b = (-3, -2, -9)^T$ , выполнить следующее:

- а. На основе метода Зейделя сформулировать метод минимальных поправок в каноническом виде.
- б. Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?
- в. Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 11

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 4 & 0 & -1 \\ 0 & 5 & 2 \\ -1 & 2 & 10 \end{pmatrix}$$

и вектор  $b = (-3, -2, -9)^T$ , выполнить следующее:

- а. На основе метода Зейделя сформулировать неявный метод скорейшего спуска в каноническом виде.
- б. Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?

- в.** Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 12

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 2 & 0 \\ 2 & 5 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

и вектор  $b = (8, -4, 3)^T$ , выполнить следующее:

- а.** На основе метода Зейделя сформулировать метод минимальных поправок в каноническом виде.
- б.** Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?
- в.** Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 13

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 10 & 2 & 0 \\ 2 & 5 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

и вектор  $b = (8, -4, 3)^T$ , выполнить следующее:

- а.** На основе метода Зейделя сформулировать неявный метод скорейшего спуска в каноническом виде.
- б.** Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т.е.  $x^0 = (0, 0, 0)^T$ ?

- в. Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

### Задача 14

Для системы алгебраических уравнений вида

$$Ax = b,$$

где матрица

$$A = \begin{pmatrix} 5 & -1 & 0 \\ -1 & 4 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

и вектор  $b = (4, 2, -1)^T$ , выполнить следующее:

- а. На основе метода Зейделя сформулировать неявный метод скорейшего спуска в каноническом виде.
- б. Определить оптимальный параметр  $\tau_1$  для нулевого начального приближения, т. е.  $x^0 = (0, 0, 0)^T$ ?
- в. Вычислить одну итерацию и найти апостериорную оценку ошибки в норме  $\|\cdot\|_\infty = \max_{i=1,2,3} \{|x_i|\}, \quad x \in \mathbb{R}^3$ .

## 9.15 Заключение по разделу 9

В данном разделе рассмотрены алгоритмы решения систем линейных алгебраических уравнений иного – по сравнению с предшествующими разделами – класса, а именно, итерационные, т. е., приближённые методы решения СЛАУ.

В проекте № 6 предлагается выполнить программирование семи различных итерационных методов и на этой основе решить две задачи вычислительной линейной алгебры: (1) найти решение СЛАУ и (2) найти обратную матрицу. При этом матрицу системы предлагается формировать двумя различными способами: вводить «вручную» с клавиатуры компьютера или генерировать случайным образом из числа положительно определённых матриц.

Этот проект кардинально отличается по применяемым в нём методам от всех предшествующих проектов этого учебного пособия. Он может рассматриваться как (дополнительный) проект повышенной сложности для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

IV

СПЕЦИАЛЬНЫЙ КУРС



# 10

## Проект № 7 «Разреженные формы *LU*-разложения»

### 10.1 Упакованные формы хранения матриц

Для решения систем линейных алгебраических уравнений с разреженными матрицами используются те же самые методы гауссова исключения, что и для линейных систем с заполненными матрицами. Отличие состоит только в выборе главного элемента и в способе хранения матрицы коэффициентов системы уравнений [11].

Так как разреженные матрицы имеют небольшое число ненулевых элементов, в целях экономии оперативной памяти ЭВМ такие матрицы хранят в *упакованном виде*. Рассмотрим четыре наиболее употребимых способа упаковки, используемых для хранения произвольных разреженных матриц.

**Пример 10.1.** В качестве примера возьмём квадратную матрицу  $A$  порядка 6 с тринадцатью ненулевыми элементами:  $a_{11} = 1$ ,  $a_{13} = 3$ ,  $a_{14} = -2$ ,  $a_{21} = 1$ ,  $a_{25} = 5$ ,  $a_{33} = 7$ ,  $a_{34} = 2$ ,  $a_{42} = -3$ ,  $a_{46} = -1$ ,  $a_{51} = 1$ ,  $a_{54} = 3$ ,  $a_{65} = 2$ ,  $a_{66} = 2$ .

В излагаемых ниже схемах хранения разреженной матрицы  $A$  упаковка осуществляется по строкам.

**Схема 1.** Каждому ненулевому элементу матрицы  $A$  ставится в соответствие запись, состоящая из двух полей. Первое поле записи содержит номер столбца, а второе – значение элемента. Нуль во втором поле означает начало новой строки. В этом случае первое поле содержит номер новой строки. Нули в обоих полях записи указывают на конец массива, хранящего данную разреженную матрицу  $A$ . В соответствии с этой схемой матрица  $A$  примера 10.1 будет храниться в виде следующего массива:

1	0	1	1.0	3	3.0	4	-2.0	2	0	1	1.0	5	5.0	⇒
3	0	3	7.0	4	2.0	4	0	2	-3.0	6	-1.0	5	0	⇒
1	1.0	4	3.0	6	0	5	2.0	6	2.0	0	0			

**Схема 2.** Информация о матрице хранится в трёх массивах. В массиве **a** хранятся ненулевые элементы матрицы  $A$ . В массиве **b** хранятся индексы столбцов, а в массиве **c** – указатели индексов строк, т. е., на  $k$ -м месте в массиве **c** хранится местоположение первого ненулевого элемента  $k$ -й строки в массиве **a**. В соответствии с этой схемой матрица  $A$  примера 10.1 будет храниться в виде трёх массивов

$$\begin{aligned}\mathbf{a} &= (1.0, 3.0, -2.0, 1.0, 5.0, 7.0, 2.0, -3.0, -1.0, 1.0, 3.0, 2.0, 2.0), \\ \mathbf{b} &= (1, 3, 4, 1, 5, 3, 4, 2, 6, 1, 4, 5, 6), \\ \mathbf{c} &= (1, 4, 6, 8, 10, 12).\end{aligned}$$

**Схема 3.** Каждому ненулевому элементу данной матрицы однозначно ставится в соответствие целое число вида

$$\lambda(i, j) = (i - 1)n + j, \quad a_{ij} \neq 0. \quad (10.1)$$

Хранение ненулевых элементов разреженной матрицы обеспечивается двумя массивами. В массиве **a** хранятся ненулевые элементы матрицы, в массиве **b** хранятся соответствующие им числа  $\lambda(i, j)$ . В соответствии с этой схемой матрица  $A$  примера 10.1 будет храниться в виде двух массивов:

$$\begin{aligned}\mathbf{a} &= (1.0, 3.0, -2.0, 1.0, 5.0, 7.0, 2.0, -3.0, -1.0, 1.0, 3.0, 2.0, 2.0), \\ \mathbf{b} &= (1, 3, 4, 7, 11, 15, 16, 20, 24, 25, 28, 35, 36).\end{aligned}$$

Исходная матрица по этой схеме хранения может быть восстановлена следующим образом. Сначала определяем  $i$  как такое наименьшее целое число, что

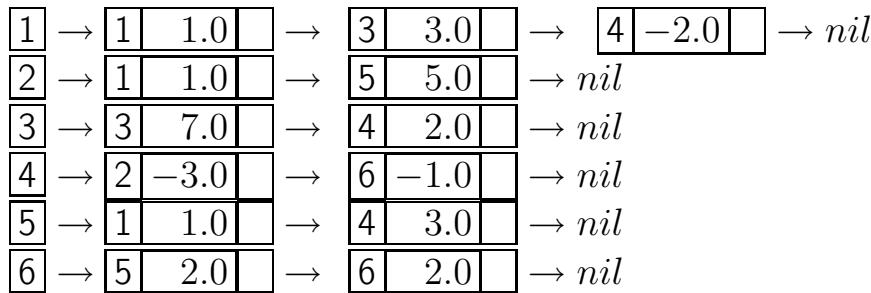
$$i \geq \lambda(i, j)/n.$$

Затем, зная  $i$ , с учётом (10.1) находим  $j$

$$j = \lambda(i, j) - (i - 1)n. \quad (10.2)$$

**Схема 4.** Для хранения каждого ненулевого элемента матрицы используется запись, состоящая из трёх полей. В первом поле хранится номер столбца, в котором стоит этот ненулевой элемент. Во втором поле хранится значение элемента, а в третьем – указатель на следующий ненулевой элемент строки

или *nil*, если это последний ненулевой элемент в строке. Таким образом, разреженная матрица хранится в виде массива указателей на списки, а каждый список содержит все ненулевые элементы одной строки. Упакованную форму матрицы  $A$  примера 10.1 в этом случае можно схематично изобразить следующим образом.



## 10.2 Выбор ведущего элемента

Способы 1–4 упаковки матриц позволяют компактно хранить матрицу  $A$  коэффициентов системы  $Ax = f$ . Однако при использовании метода Гаусса (или ему подобных) в результате модификации элементов матрицы  $A$  может значительно возрасти число ненулевых элементов. С одной стороны, это требует дополнительной памяти для хранения новых ненулевых элементов, а с другой приводит к возрастанию числа арифметических операций, что влечёт накопление ошибок округления. В связи с этим обстоятельством были предложены стратегии выбора главного элемента, позволяющие минимизировать число новых ненулевых элементов на каждом шаге метода Гаусса.

Назовём локальным заполнением на  $(k+1)$ -м шаге метода Гаусса число элементов матрицы  $A$ , которые были нулевыми после  $k$ -го шага и стали ненулевыми после  $(k+1)$ -го шага метода Гаусса. Таким образом, задача заключается в выборе в качестве главного элемента такого элемента матрицы  $A^{(k)}$ , который минимизирует локальное заполнение матрицы  $A$  на  $(k+1)$ -м шаге (как и прежде,  $A^{(k)}$  – активная подматрица матрицы  $A$ ). При этом, чем больше множество элементов, среди которых выбирается главный, тем меньше локальное заполнение. Но, с другой стороны, в качестве главного можно брать только ненулевой элемент. Поэтому вводится понятие допустимого элемента.

Допустимым элементом на  $(k+1)$ -м шаге метода Гаусса называется такой элемент активной подматрицы  $A^{(k)}$ , который удовлетворяет неравенству

$$\left| a_{ij}^{(k)} \right| > \varepsilon,$$

где  $\varepsilon$  — некоторое наперёд заданное положительное число. В лабораторном проекте № 4, описание которого дано ниже, надо взять  $\varepsilon = 10^{-3}$ , если используется тип *real*, или  $\varepsilon = 10^{-5}$ , если используется тип *extended*.

Итак, среди элементов активной подматрицы  $A^{(k)}$  в соответствии с критерием (10.2) выбирается множество допустимых элементов, а затем среди них отыскивается элемент, минимизирующий локальное заполнение на текущем шаге. Для этого используются следующие две стратегии выбора оптимального ведущего элемента.

**Стратегия I.** Локальное заполнение на  $(k+1)$ -м шаге метода Гаусса будет минимальным, если в качестве главного (ведущего) выбрать элемент  $a_{st}^{(k)}$  на позиции  $s = \alpha + k, t = \beta + k$ , где  $\alpha$  и  $\beta$  определяются из формулы

$$g_{\alpha\beta}^{(k+1)} = \min_{i,j} \{e_i^T G_{k+1} e_j\} \quad \text{для всех } |a_{i+k,j+k}^{(k)}| > \varepsilon.$$

Здесь  $G_{k+1} = B_k \bar{B}_k^T B_k$ , где  $B_k$  — матрица, полученная из  $A^{(k)}$  путем замены ненулевых элементов единицами, а  $\bar{B}_k = M - B_k$  ( $M$  — матрица, состоящая из единиц). Согласно стратегии I, в качестве главного берётся тот из числа допустимых элементов активной подматрицы  $A^{(k)}$ , которому в матрице  $G_{k+1}$  соответствует наименьший элемент.

**Стратегия II.** Локальное заполнение на  $k+1$ -м шаге метода Гаусса будет небольшим, если в качестве главного (ведущего) выбрать элемент  $a_{st}^{(k)}$ , на позиции  $s = \alpha + k, t = \beta + k$ , где  $\alpha$  и  $\beta$  определяются из формулы

$$g_{\alpha\beta}^{(k+1)} = \min_{i,j} \{e_i^T \hat{G}_{k+1} e_j\} \quad \text{для всех } |a_{i+k,j+k}^{(k)}| > \varepsilon.$$

Здесь  $\hat{G}_{k+1} = (B_k - I_{n-k})M(B_k - I_{n-k})$ , где матрицы  $M$  и  $B_k$  имеют тот же самый смысл, что и в стратегии I, а  $I_{n-k}$  обозначает единичную матрицу размера  $n - k$ . Хотя стратегия II не обеспечивает минимальное заполнение на текущем шаге, она очень просто реализуется на практике, и её применение приводит к сравнительно небольшому числу новых ненулевых элементов.

Использование упакованной формы хранения матрицы  $A$  и более сложной процедуры выбора главного элемента требует значительного увеличения затрат машинного времени, поэтому перенос процедур и функций из лабораторного проекта № 1, осуществляющих решение системы линейных алгебраических уравнений, не даст оптимальный по времени результат. Это связано с тем, что поиск  $(i, j)$ -го элемента матрицы  $A$  в упакованной форме требует существенных затрат машинного времени. Следовательно, при написании оптимальной по

времени счёта программы таких действий надо избегать. Это можно сделать, если применить метод Гаусса непосредственно к упакованной форме хранения матрицы  $A$ , т. е., только к её ненулевым элементам.

Поскольку стандартный метод Гаусса (см. подразд. 7.1) оперирует со строками матрицы  $A$ , разреженные матрицы удобно хранить по строкам. Это позволит эффективно организовать такие операции, как нормировка строки, умножение строки на число, вычитание строки, потому что в этом случае все ненулевые элементы каждой строки матрицы  $A$  в упакованной форме хранения образуют непрерывную последовательность. Следовательно, операции нормировки, умножения и вычитания строки можно проводить только для ненулевых элементов.

Такой подход подразумевает, что ненулевые элементы матрицы  $A$  модифицируются в том порядке, в котором они хранятся в упакованной форме, что исключает затраты на поиск нужного элемента. Модификация происходит следующим образом. Берётся очередной ненулевой элемент матрицы  $A$ . Определяется его местоположение в матрице, т. е., индексы  $i$  и  $j$ . Затем с помощью дополнительных массивов обратных перестановок определяется местоположение этого элемента в матрице с переставленными столбцами и строками. Если в результате этот элемент лежит в активной подматрице, то он изменяется по известным формулам метода Гаусса. Здесь надо предусмотреть процедуру вставки элемента на случай появления нового ненулевого элемента и процедуру удаления элемента из упакованной формы, если ненулевой элемент стал нулевым. затем обрабатывается следующий ненулевой элемент матрицы  $A$ .

Оптимизация по времени процедуры решения системы линейных алгебраических уравнений позволяет значительно повысить эффективность программы. Однако этого недостаточно. Оптимальной по быстродействию будет лишь та программа, где дополнительно оптимизирована процедура выбора главного элемента. Во-первых, надо подумать над эффективным вычислением элементов матрицы  $G_k$  (или  $\widehat{G}_k$ ). Во-вторых, надо организовать вычисление этой матрицы так, чтобы исключить поиск элементов в упакованной форме. Только учёт всех этих особенностей решения систем линейных алгебраических уравнений с разреженной матрицей коэффициентов позволит написать эффективную по затратам машинного времени программу.

### 10.3 Задание на лабораторный проект № 7

Написать и отладить программу, реализующую заданный вариант метода исключения с заданной схемой хранения разреженных матриц и заданной стратегией выбора главного элемента, для численного решения систем вида  $Ax = f$ . Отделить основные части программы:

- а) подпрограмму упаковки матрицы  $A$ ;
- б) подпрограмму метода исключения;
- в) подпрограмму выбора главного элемента;
- г) сервисные подпрограммы.

Уделить особое внимание эффективности программы (в смысле скорости счёта). Программа должна решать систему линейных алгебраических уравнений порядка  $n = 200$  не более чем за 3 минуты (для персонального компьютера 486 АТ с сопроцессором). Предусмотреть пошаговое выполнение алгоритма исключения с выводом результата на экран.

Выполнить следующие пункты задания:

1. Для заданной матрицы  $A$  выдать на экран упакованную форму в соответствии со своим вариантом, построить таблицу зависимости оценочного и реального локального заполнения от номера шага исключения (для этого предусмотреть ввод матрицы с экрана).

2. Оценить точность решения систем линейных алгебраических уравнений, имеющих порядок  $n$  от 100 до 200 (через 5). Для этого сгенерировать случайные матрицы  $A$  (не более 10 ненулевых элементов в строке), выбрать  $x^*$  – точное решение и образовать правые части  $f = Ax^*$ . Провести анализ точности решения как функцию от  $n$ . Результаты вывести в таблицу и на график.

Для случайного заполнения матрицы  $A$  использовать алгоритм:

(а) Ненулевыми целыми числами, выбранными случайным образом из интервала  $[-100; 100]$ , заполнить обратную диагональ матрицы  $A$ .

(б) В каждой строке случайным образом выбрать количество ненулевых элементов (от 1 до 10 с учётом элементов по пункту (а)), их местоположение (номер столбца от 1 до  $n$ ) и значение (ненулевые целые числа, лежащие в интервале от  $-100$  до  $100$ ).

В качестве точного решения взять вектор  $x^* = (1, 2, \dots, n)$ . Если при решении системы  $Ax = f$  выяснится, что матрица  $A$  вырождена (плохо обусловлена), сгенерировать новую матрицу того же порядка и решить систему линейных алгебраических уравнений с новой матрицей  $A$  и новой правой частью. Для оценки точности решения использовать норму вектора по формуле (8.5), с 345.

3. Определить скорость решения систем из пункта 2. Результаты вывести в таблицу и на график.

4. Системы из пункта 2 решить методом исключения переменных двумя способами: способ 1 – из лабораторного проекта № 1, способ 2 – из лабораторного проекта № 4 (в соответствии со своим вариантом по лабораторному проекту № 1 или № 4). В этом случае разреженная матрица должна размещаться в памяти ЭВМ полностью (в распакованном виде). Сравнить точность решения и затраты машинного времени, получаемые, с одной стороны, в лабораторном проекте № 1 (или № 4) и, с другой стороны, в лабораторном проекте № 6.

**Замечание 10.1.** По ходу проведения численных экспериментов на экран дисплея должны выводиться таблицы следующего вида.

### Решение систем линейных алгебраических уравнений

Порядок матрицы	Время		Точность	
	Заполненная матрица	Разреженная матрица	Заполненная матрица	Разреженная матрица

**Замечание 10.2.** Некоторые результаты экспериментов необходимо сохранять в текстовый файл, чтобы затем вывести на экран в виде графиков.

### Графики решения систем линейных алгебраических уравнений:

- зависимость точности решения от порядка матриц для способа 1 решения (см. п. 4);
- зависимость точности решения от порядка матриц для способа 2 решения (см. п. 4);
- зависимость времени решения от порядка матриц для способа 1 решения (см. п. 4);
- зависимость времени решения от порядка матриц для способа 2 решения (см. п. 4).

Таблица 10.1. Варианты задания на лабораторный проект № 6

Вид разложения	Стратегия I				Стратегия II			
	a	b	c	d	a	b	c	d
$A = \bar{L}U$	1	2	3	4	5	6	7	8
$A = L\bar{U}$	9	10	11	12	13	14	15	16
$A = \bar{U}L$	17	18	19	20	21	22	23	24
$A = U\bar{L}$	25	26	27	28	29	30	31	32
$A = L\bar{U}$ в виде $L, \bar{U}^{-1}$	33	34	35	36	37	38	39	40
$A = U\bar{L}$ в виде $\bar{L}^{-1}, U$	41	42	43	44	45	46	47	48

<sup>a</sup> – схема 1 хранения разреженной матрицы  $A$ ;<sup>b</sup> – схема 2 хранения разреженной матрицы  $A$ ;<sup>c</sup> – схема 3 хранения разреженной матрицы  $A$ ;<sup>d</sup> – схема 4 хранения разреженной матрицы  $A$ .

## 10.4 Варианты задания на лабораторный проект № 7

Студент определяет номер своего варианта по табл. 10.1 (см. выше) соответственно своему порядковому номеру в списке учебной группы. В таблице приведены 48 номеров вариантов задания на лабораторный проект № 6. Все варианты различаются по следующим признакам:

- стратегии I или II выбора главного элемента;
- четыре схемы упаковки и хранения разреженной матрицы  $A$ ;
- шесть вариантов метода исключения, определяемых видом разложения матрицы.

## 10.5 Заключение по разделу 10

В данном разделе рассмотрены стандартные алгоритмы  $LU$ -разложения матрицы, численно эквивалентные методу Гаусса последовательного исключения неизвестных в системе линейных алгебраических уравнений, но применительно к особому виду матриц – разреженным матрицам. В таких матрицах много нулевых элементов, и их хранить в памяти не надо. Однако в процессе вычислений, например, по методу  $LU$ -разложения, нулевые элементы

то исчезают, то появляются в других местах матрицы. Это создаёт дополнительные сложности в схеме доступа к элементам в процессе вычислений.

В проекте № 7 предлагается выполнить программирование *LU*-разложения для разреженных матриц и на этой основе решить основную задачу вычислительной линейной алгебры – найти решение СЛАУ. При этом требуется сравнить по быстродействию традиционный метод лабораторного проекта № 1 и метод для разреженных матриц.

Этот проект является специальным для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

# 11

## Проект № 8 «Одновременные наименьшие квадраты»

### 11.1 Линейная задача наименьших квадратов

Во многих приложениях, связанных с обработкой экспериментальных данных, необходимо отыскивать такой вектор  $x \in \mathbb{R}^n$ , линейные комбинации компонент которого,  $Ax$ , где  $A = A(m, n)$  – матрица размера  $(m \times n)$ , как можно более близки или, ещё лучше, равны данным значениям, образующим вектор  $z \in \mathbb{R}^m$ , т. е.  $Ax \approx z$ . Если мерой близости двух векторов считать квадрат евклидовой нормы разностного вектора, в данном случае, вектора  $v \triangleq z - Ax$ , то указанная задача есть линейная задача о наименьших квадратах.

Возможность сделать равным нулю вектор невязок  $v = z - Ax$  существует тогда и только тогда, когда  $z \in \mathcal{R}(A)$ , где  $\mathcal{R}(A)$  – пространство столбцов матрицы  $A$ . В этом случае имеем совместную систему уравнений  $Ax = z$ . Однако  $z$  – вектор наблюдений, то есть экспериментальных данных и  $A$  – матрица, которую задают до того, как получат  $z$  и которую в различных приложениях называют либо матрицей регрессоров, либо матрицей наблюдений, либо матрицей плана эксперимента. Совсем не обязательно, что условие  $z \in \mathcal{R}(A)$  будет выполнено, например, из-за случайных погрешностей  $v$  во время регистрации экспериментальных данных. Тогда

$$z = Ax + v, \quad (11.1)$$

и решение по методу наименьших квадратов (для краткости, *MНK-решение*) есть вектор  $\bar{x}$ , доставляющий минимум функционалу качества (см. рис. 11.1):

$$J(x) = (z - Ax)^T(z - Ax) = \sum_{j=1}^m v(j)^2 = v^T v. \quad (11.2)$$

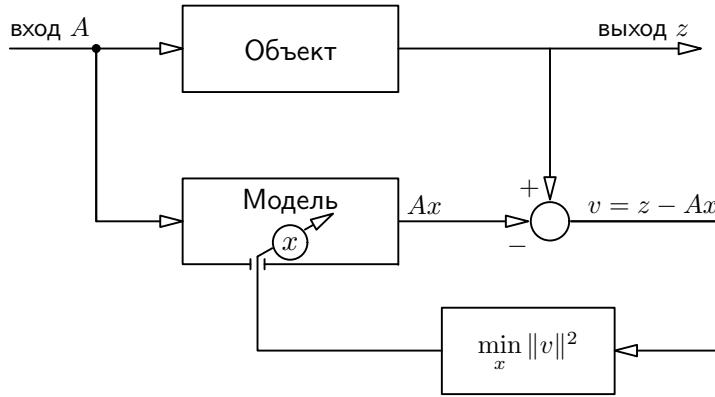


Рис. 11.1. Линейная задача наименьших квадратов

Требуя минимум этого критерия, для искомого  $\bar{x}$  получаем так называемые *нормальные уравнения*:

$$A^T A \bar{x} = A^T z. \quad (11.3)$$

Их решение всегда существует (обе части равенства (11.3) принадлежат одному и тому же пространству  $\mathcal{R}(A^T)$  столбцов матрицы  $A^T$ ), но может быть не единственным (если  $\text{rank } A < n$ ). В последнем случае из всех  $\bar{x}$  выбирают то единственное,  $\bar{x}_0$ , которое имеет минимальную норму  $\|\bar{x}_0\|$ . Этот вектор называют *нормальным псевдорешением*. Известно (см. [6]), что

$$\bar{x}_0 = A^+ z, \quad (11.4)$$

где  $A^+$  – *псевдообратная матрица* к  $A$ . Как уже отмечалось, в качестве определения  $A^+$  применяют различные формулировки. Здесь для этого используем геометрический подход:  $A^+$  есть такая матрица в выражении (11.4), что для любого  $z \in \mathbb{R}^m$  вектор  $\bar{x}_0 \in \mathbb{R}^n$  удовлетворяет двум условиям:

$$A \bar{x}_0 = p, \quad p \in \mathcal{R}(A), \quad z - p \perp \mathcal{R}(A). \quad (11.5)$$

$$\bar{x}_0 \in \mathcal{R}(A^T). \quad (11.6)$$

Условие (11.5) требует, чтобы  $\bar{x}_0$  отвечал совместной системе  $A \bar{x}_0 = p$ , где  $p$  – проекция вектора  $z$  на  $\mathcal{R}(A)$ , а условие (11.6) требует, чтобы этот  $\bar{x}_0$  был взят из пространства  $\mathcal{R}(A^T)$  строк матрицы  $A$ . Условие (11.5), таким образом, выбирает  $\bar{x}_0 = \bar{x}$ , чтобы минимизировать функционал (11.2), а условие (11.6) среди всех таких  $\bar{x}$  выбирает единственный  $\bar{x}_0$  с минимальной нормой.

Часто матрицу  $A$  выбирают так, чтобы она имела полный столбцовый ранг,  $\text{rank } A = n$ . Тогда  $m \geq n$ ,  $\bar{x}$  единственны и равно  $\bar{x}_0$ ,  $A^+ = (A^T A)^{-1} A^T$  и

$$\bar{x}_0 = (A^T A)^{-1} A^T z. \quad (11.8)$$

Однако иногда такое условие не выполняется, и тогда  $\bar{x}_0 = A^+z$ , где  $A^+$  – псевдообратная матрица.

## 11.2 Метод нормальных уравнений

Нормальные уравнения (11.3) показаны выше (см. стр. 389). Довольно часто их используют для отыскания МНК-решения  $\bar{x}$  [19]. Этот подход противоположен последовательным методам, берущим начало от идеи расщепления исходной переопределённой системы на априорную и текущую части (см. подразд. 12.3). Поэтому иногда его называют «одновременным решением» всех нормальных уравнений, характеризующих всю исходную переопределённую систему  $Ax \approx z$  (см. стр. 388 и выражение (11.1)).

**Метод нормальных уравнений.** Алгоритм использует полный вектор наблюдений  $z \in \mathbb{R}^m$  и всю матрицу плана эксперимента  $A \in \mathbb{R}^{m \times n}$ , имеющую  $\text{rank}(A) = n$ . По матрице  $A$  и вектору  $z$  алгоритм позволяет найти решение  $\bar{x}$  задачи наименьших квадратов, доставляющее  $\min \|z - Ax\|^2$ .

Алгоритм:

1. Вычисляют нижний треугольник матрицы  $\Lambda = A^T A$ .
2. Вычисляют  $d = A^T z$ .
3. Вычисляют разложение Холесского  $\Lambda = SS^T$ .
4. Решают сначала систему  $Sy = d$  и затем систему  $S^T \bar{x} = y$ .

Здесь в качестве разложения Холесского может быть взято либо нижнее треугольное разложение ( $S = L$ ), либо верхнее треугольное разложение (при  $S = U$ ) – см. подразд. 4.2, стр. 136. Оба эти варианта требуют многократного применения операции извлечения квадратного корня. Если в п. 3 алгоритма вычислять разложение Холесского без операции квадратного корня: либо  $\Lambda = \bar{S}D\bar{S}^T$  с  $\bar{S} = \bar{L}$ , либо  $\bar{S} = \bar{U}$ , то п. 4 заменится на последовательное решение трёх систем:  $\bar{S}w = d \Rightarrow Dy = w \Rightarrow \bar{S}^T \bar{x} = y$ .

## 11.3 Формирование матрицы $A$

Используйте следующие четыре варианта формирования матрицы  $A$  для оценки скорости и точности двух методов решения переопределённых систем линейных алгебраических уравнений, как указано в пп. В и Г задания (см. задание ниже в подразд. 11.4).

Для  $n = 2$  до  $12$  с шагом  $5$  выполнять

Для  $m = n + 1$  до  $26$  с шагом  $3$  выполнять

Для  $i = 1$  до  $m$  выполнять

Для  $j = 1$  до  $n$  выполнять

$$\text{Вариант 1: } a_{ij} = \sin\left(\frac{(i-1)j}{m}\right),$$

$$\text{Вариант 2: } a_{ij} = \frac{1}{1 + 66 \left[ \frac{(i-1)j}{m} \right]^4},$$

$$\text{Вариант 3: } a_{ij} = 1/(i+j),$$

$$\text{Вариант 4: } a_{ij} = 100(RAN - 1/2),$$

где  $RAN$  – равномерно распределённая в интервале  $[0, 1]$  случайная величина, получаемая независимо для каждого элемента  $a_{ij}$  матрицы  $A$ .

## 11.4 Задание на лабораторный проект № 8

**А.** Спроектировать и отладить подпрограмму решения несовместной системы  $Ax = z$ ,  $A = A(m, n)$ ,  $m > n$ ,  $\text{rank}(A) = n$ , в смысле наименьших квадратов при помощи заданного метода ортогонального приведения. Обосновать проект и дать набор инструкций для пользователей подпрограммы. Сделать подсчёт операций (отдельно сложения, умножения, деления и извлечения квадратного корня) в зависимости от  $m$  и  $n$ , где  $m$  – число строк матрицы  $A$ , а  $n$  – число столбцов. Рекомендуется в качестве основы вашего проекта использовать ту программу, которая была вами написана и отлажена в рамках лабораторного проекта № 3 для решения совместной системы уравнений  $Ax = f$  с квадратной матрицей  $A$  методом ортогонального приведения. Для этого в указанной программе достаточно осуществить небольшие изменения.

**Б.** Повторить п. А задания на основе метода нормальных уравнений  $A^T A \bar{x} = A^T z$ , которым удовлетворяет искомое решение  $\bar{x}$  в смысле наименьших квадратов, называемое нормальным псевдорешением несовместной системы  $Ax \approx z$ . Для этого применить вашу программу решения системы уравнений с симметричной положительно определенной матрицей методом квадратного корня (разложение Холесского) из лабораторной работы № 2.

**В.** Спроектировать и провести вычислительный эксперимент для сравнения скорости выполнения двух программ по пп. А. и Б. Использовать четыре различных варианта генерации  $n$  векторов длины  $m$  для формирования матрицы  $A$

(см. подразд. 11.3) при  $2 \leq n \leq 12$ ,  $n+1 \leq m \leq 26$ . Результаты представить в виде таблиц и графиков, которые иллюстрируют поведение каждого метода на каждом варианте генерации матрицы  $A$ . Дать обобщённую (по вариантам матрицы) картину зависимости времени выполнения от значений параметров  $t$  и  $n$  матрицы. Проанализировать соотношение между фактическим временем выполнения и числом операций, рассчитанным по пп. А и Б. Правые части уравнений формировать, как указано в следующем пункте задания.

**Г.** Подобно п. В, сравнить точность нахождения нормального псевдорешения переопределённой системы линейных уравнений для методов из пп. А и Б. Для этого также четырьмя способами сгенерировать матрицу  $A$  (см. подразд. 11.3), выбрать (принудительно задать) точное нормальное псевдорешение  $x^*$  и образовать вектор  $z^* = Ax^*$ . К элементам этого вектора добавить случайные числа  $v_i$ , чтобы образовать правые части  $z_i = z_i^* + av_i$ ,  $i = 1, 2, \dots, m$ . Написать подпрограмму генерации псевдослучайных чисел  $v_i$  так, чтобы каждое  $v_i$ , имело стандартное нормальное распределение (с нулевым средним значением и единичной дисперсией). При этом любые случайные величины  $v_i$ ,  $v_j$  ( $i \neq j$ ) должны моделироваться как попарно независимые. Предусмотреть множитель-переключатель  $a$ , чтобы по желанию включать или отключать добавление случайных чисел  $v_i$ , или же регулировать их уровень.

В качестве точного нормального псевдорешения взять вектор  $x^* = [1, 2, \dots, m]^T$ . Для оценки точности оценивания использовать норму вектора

$$\|x\|_\infty = \max_i(|x_i|).$$

При использовании программы, где выполняется ортогональное приведение  $QA = B$ , для проверки правильности метода убедиться в справедливости равенства  $A^T A - B^T B = 0$ . Для этого использовать норму матрицы

$$\|A\|_\infty = \max_i \left( \sum_{j=1}^n |a_{ij}| \right).$$

**Д.** Представить обобщённую аттестацию двух подпрограмм и соответствующих методов, основанную на проведённых наблюдениях. Обсудить любые сравнительные достоинства и недостатки, поддающиеся количественной оценке, и предложить план дальнейших вычислительных экспериментов, которые могли бы помочь уточнить различия между рассмотренными выше методами решения переопределённых систем уравнений.

**E.** Решить следующую прикладную задачу [14]. Для  $i = 1, 2, \dots, m$  ( $m$  кратно четырём) имеем

$$y_i = x_1 w_i + x_2 w_{i-1}, \quad w_i = \sin(2\pi i/m), \quad d_i = 2 \cos(2\pi i/m).$$

Найти оптимальное значение  $\bar{x} = (\bar{x}_1, \bar{x}_2)^T$  вектора коэффициентов  $x = (x_1, x_2)^T$ , доставляющее минимум средней квадратической ошибке

$$J(x) = \frac{1}{m} \sum_{i=1}^m (y_i - d_i)^2.$$

Решение выполнить двумя способами: аналитически и численно. Аналитическое решение должно включать:

1) эквивалентную постановку задачи решения переопределённой системы  $Ax \approx z$ ,

2) решение для неё нормальных уравнений, дающее

$$\bar{x} = 2 \begin{bmatrix} \operatorname{ctg}(2\pi/m) & | & -\operatorname{cosec}(2\pi/m) \end{bmatrix}^T,$$

3) представление критерия качества для общего случая в виде

$$J(x) = J_{\min} + (x - \bar{x})^T \Lambda (x - \bar{x}),$$

где  $\Lambda = A^T A$  – информационная матрица,  $\bar{x} = (A^T A)^{-1} A^T z$  – нормальное псевдорешение,

4) доказательство того, что в данном конкретном случае

$$J_{\min} = \min_x (J(x)) = J(\bar{x}) = 0,$$

5) вычисление собственных значений  $(\lambda_1, \lambda_2)$  матрицы  $\Lambda$ , дающее

$$\lambda_1 = \frac{1}{2} \left[ 1 - \cos(2\pi/m) \right], \quad \lambda_2 = \frac{1}{2} \left[ 1 + \cos(2\pi/m) \right],$$

6) вычисление соответствующих собственных векторов  $(v_1, v_2)$  матрицы  $\Lambda$ ,

7) представление критерия качества для общего случая в виде

$$J(x) = J_{\min} + e^T Q \bar{\Lambda} Q^T e,$$

где  $e = x - \bar{x}$  – отклонение  $x$  от оптимального значения  $\bar{x}$ ,  $Q$  – матрица ортонормированных собственных векторов матрицы  $\Lambda$ , диагональная матрица  $\bar{\Lambda} = \operatorname{diag} [\lambda_1, \lambda_2]$  составлена из собственных значений матрицы  $\Lambda$ , так что  $\Lambda = Q \bar{\Lambda} Q^{-1} = Q \bar{\Lambda} Q^T$ .

Таблица 11.1. Варианты задания на лабораторный проект № 8

Вариант заполнения матрицы $R$	Отражения Хаусхолдера		Вращения Гивенса		Ортогонализация Грама-Шмидта		
	a	b	a	b	c	d	e
 $\triangleq R_{ne}$	1	2	3	4	5	6	7
 $\triangleq R_{nw}$	8	9	10	11	12	13	14
 $\triangleq R_{se}$	15	16	17	18	19	20	21
 $\triangleq R_{sw}$	22	23	24	25	26	27	28

<sup>a</sup> – столбцово-ориентированный алгоритм;

<sup>b</sup> – строчно-ориентированный алгоритм;

<sup>c</sup> – классическая схема;

<sup>d</sup> – модифицированная схема;

<sup>e</sup> – модифицированная схема с выбором ведущего вектора.

Изобразите на экране в системе координат  $[x_1, x_2] = x^T$  линии постоянных уровней критерия  $J(x) = \text{const}$  для шести значений  $\text{const} = \{1, 2, 3, 4, 5, 6\}$  в окрестности точки минимума критерия для одного из значений  $m = 4, 8, 12, 16, 20, 24, 28$  или  $32$  (по выбору). Объяснить геометрический смысл матриц  $Q$  и  $\bar{\Lambda}$  в последнем представлении критерия.

Численное решение должно включать вычисление решения  $\bar{x}$  с помощью двух методов (по пп. А и Б) и сравнительную оценку точности двух решений для нормы вектора  $\|\bar{x}\|_\infty$  (в зависимости от  $m = 4, 8, 12, 16, 20, 24, 28, 32$ ). Эту зависимость необходимо представить таблицей и графиком.

## 11.5 Варианты задания на лабораторный проект № 8

По теме «Одновременное решение нормальных уравнений» студентам предлагается выполнение лабораторной работы – проекта № 8.

Задание на этот проект содержит 28 вариантов, которые аналогичны вариантам, приведённым в табл. 5.2 (см. с. 199) для проекта № 3. Все варианты различаются по следующим признакам:

- четыре варианта заполнения треугольной матрицы  $R$ ;
- три вида ортогональных преобразований:
  - 1) отражения Хаусхолдера,
  - 2) вращения Гивенса,
  - 3) ортогонализация Грама–Шмидта,
- две разновидности алгоритма ортогонализации по методу Хаусхолдера и по методу Гивенса:
  - 1) столбцово-ориентированный алгоритм,
  - 2) строчно-ориентированный алгоритм,
- три разновидности ортогонализации по методу Грама–Шмидта:
  - 1) классическая схема,
  - 2) модифицированная схема,
  - 3) модифицированная схема с выбором ведущего вектора.

Если нет других указаний преподавателя, выбирайте ваш вариант в табл. 11.1 на с. 394 по вашему номеру в журнале студенческой группы.

## 11.6 Заключение по разделу 11

В данном разделе рассмотрен классический метод наименьших квадратов для решения несовместных (переопределённых) систем линейных алгебраических уравнений. Он назван «классическим», потому что появился первым и давно известен. Он сводится к решению так называемой нормальной системы уравнений, при этом все уравнения этой системы решаются одновременно.

В проекте № 8 предлагается выполнить программирование одновременного метода нормальных уравнений. При этом решение должно отыскиваться на основе ортогональных преобразований: метода отражений (Хаусхолдера), метода плоских вращений (Гивенса), либо метода ортогонализации Грама–Шмидта.

Этот проект является специальным для освоения студентами дисциплин «Вычислительная математика» или «Численные методы» и подготовительным к заключительному проекту № 9 в следующем разделе.

# 12

## Проект № 9 «Рекуррентные наименьшие квадраты»

### 12.1 Статистическая интерпретация

Предположим, что вектор ошибок  $v$  в уравнении (11.1) образован из случайных величин с нулевым средним и известной матрицей ковариации

$$\mathbf{E}\{v\} = 0, \quad \mathbf{E}\{vv^T\} = P_v, \quad (12.1)$$

где  $\mathbf{E}\{\cdot\}$  — оператор математического ожидания (среднего) над  $\cdot$ , и  $P_v$  — ПО (положительно определённая) матрица. Найдём квадратно-корневое разложение  $P_v = SS^T$  (например, разложение Холесского). Если теперь умножить вектор  $z$  (11.1) на  $S^{-1}$ , то данные  $\bar{z} = S^{-1}z$  получают представление

$$\bar{z} = \bar{A}x + \bar{v} \quad (12.2)$$

с матрицей  $\bar{A} = S^{-1}A$  и ошибками  $\bar{v} = S^{-1}v$ . Этот вектор ошибок всегда имеет единичную ковариацию:

$$\mathbf{E}\{\bar{v}\bar{v}^T\} = \mathbf{E}\{S^{-1}vv^TS^{-T}\} = S^{-1}\mathbf{E}\{vv^T\}S^{-T} = S^{-1}SS^TS^{-T} = I_m,$$

где  $I_m$  — единичная матрица размера  $(m \times m)$ . Вследствие этого данные  $\bar{z}$  называют *нормализованными экспериментальными данными*. Значение представления (12.2) заключается в том, что оно демонстрирует, как сконструировать вектор некоррелированных между собой измерений с единичной дисперсией из вектора, элементы которого произвольно взаимно коррелированы (декоррелировать и нормализовать его). Ниже предполагаем, что данные  $z$  (11.1) уже декоррелированы и нормализованы, так что

$$\mathbf{E}\{v\} = 0, \quad \mathbf{E}\{vv^T\} = I_m, \quad (12.3)$$

где  $I_m$  — единичная матрица размера  $(m \times m)$ . При этом из (11.3) находим

$$A^T A \bar{x} = A^T z = A^T A x + A^T v,$$

$$A^T A (\bar{x} - x) = A^T v.$$

Отсюда, если  $\det(A^T A) \neq 0$ , имеем

$$\mathbf{E}\{\bar{x}\} = x, \quad (12.4)$$

$$(A^T A) \mathbf{E}\{(\bar{x} - x)(\bar{x} - x)^T\} (A^T A) = A^T \mathbf{E}\{vv^T\} A = A^T A. \quad (12.5)$$

Соотношение (12.4) выражает собой *свойство несмещенности решения* (оценки)  $\bar{x}$  относительно неизвестного (постоянного) вектора  $x$ , измеряемого в виде экспериментальных данных  $z$ , (11.1) или (12.2). Соотношение (12.5) даёт выражение для ковариации оценки  $\bar{x}$  в виде

$$P_{\bar{x}} = \mathbf{E}\{(\bar{x} - x)(\bar{x} - x)^T\} = (A^T A)^{-1}. \quad (12.6)$$

при определении  $\bar{x}$  по нормализованным экспериментальным данным.

Обратная матрица  $P_{\bar{x}}^{-1}$  от ковариации  $P_{\bar{x}}$  называется *информационной матрицей*. Её обозначение будет  $\Lambda_{\bar{x}}$  или просто  $\Lambda$ . При использовании нормализованных данных она равна  $A^T A$ , а в более общем случае (12.1) она равна  $\Lambda = A^T P_v^{-1} A$ .

## 12.2 Включение априорных статистических данных

Предположим, что в добавление к линейной системе (11.1) мы имеем априорную несмешённую оценку неизвестного вектора  $x$  в виде  $\tilde{x}$  и соответствующую априорную информационную матрицу  $\tilde{\Lambda}$ . Это означает, что  $\mathbf{E}\{\tilde{x}\} = x$  и

$$\tilde{\Lambda}^{-1} = \mathbf{E}\{(\bar{x} - x)(\bar{x} - x)^T\} = \tilde{P}, \quad (12.7)$$

где  $\tilde{P}$  — ковариация оценки  $\tilde{x}$ . Найдём какой-нибудь квадратный корень  $\tilde{\Lambda}^{1/2}$  из матрицы  $\tilde{\Lambda}$ , например, по одному из разложений Холлесского (см. подразд. 4.2):

$$\tilde{\Lambda} = \tilde{\Lambda}^{1/2} \tilde{\Lambda}^{T/2} = \tilde{R}^T \tilde{R},$$

где  $\tilde{\Lambda}^{1/2} = \tilde{R}^T$ . Образуем вектор  $\tilde{v} = (\tilde{\Lambda}^{1/2})^T (\tilde{x} - x) = \tilde{R}(\tilde{x} - x)$ . Этот вектор имеет смысл *нормализованной ошибки* для априорной оценки  $\tilde{x}$  вектора  $x$ . Действительно, его ковариация равна единичной матрице размера  $(n \times n)$ :

$$\mathbf{E}\{\tilde{v}\tilde{v}^T\} = \tilde{R} \mathbf{E}\{(\tilde{x} - x)(\tilde{x} - x)^T\} \tilde{R}^T = \tilde{\Lambda}^{T/2} \tilde{\Lambda}^{-1} \tilde{\Lambda}^{1/2} = I_n.$$

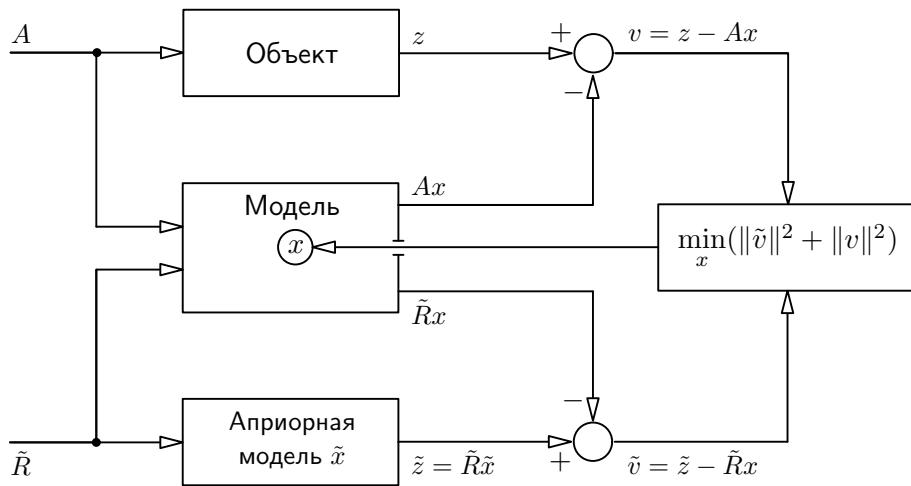


Рис. 12.1. Включение априорных данных в линейную задачу НК

Так как о векторе  $x$ , кроме экспериментальных данных  $z$ , (11.1), известна априорная оценка  $\tilde{x}$  с ковариацией  $\tilde{P} = \tilde{\Lambda}^{-1}$ , эту информацию целесообразно включить в контекст задачи о наименьших квадратах, рассматривая модифицированный функционал качества  $J_1(x) = \tilde{v}^T \tilde{v} + v^T v$  вместо (11.2). Он соединяет в себе квадрат нормы нормализованной ошибки (невязки) априорной оценки

$$\tilde{v} = \tilde{R}(\tilde{x} - x) = \tilde{\Lambda}^{T/2}(\tilde{x} - x),$$

с квадратом нормы нормализованной ошибки (невязки) экспериментальных данных  $v = z - Ax$ . Так как

$$\begin{aligned} J_1(x) &= (\tilde{x} - x)^T \tilde{\Lambda}(\tilde{x} - x) + (z - Ax)^T(z - Ax) = \\ &= (\tilde{z} - \tilde{R}x)^T(\tilde{z} - \tilde{R}x) + (z - Ax)^T(z - Ax), \end{aligned} \quad (12.8)$$

где  $\tilde{z} = \tilde{R}\tilde{x}$ , то  $J_1(x)$  может быть интерпретирован просто как критерий качества метода наименьших квадратов применительно к расширенной системе (рис. 12.1)

$$\begin{bmatrix} \tilde{z} \\ z \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ A \end{bmatrix} x + \begin{bmatrix} \tilde{v} \\ v \end{bmatrix}, \quad (12.9)$$

включающей, помимо текущих экспериментальных данных  $z$ , «дополнительные» экспериментальные данные  $\tilde{z}$ , соответствующие имеющейся в наличии априорной информации  $\tilde{x}$ ,  $\tilde{\Lambda}$ .

Обозначим через  $\hat{x}$  МНК-решение расширенной системы (12.9), доставляющее минимум функционалу (12.8). Из этого критерия для  $\hat{x}$  получаем, анало-

гично (11.3), нормальные уравнения

$$(\tilde{\Lambda} + A^T A) \hat{x} = \tilde{\Lambda} \tilde{x} + A^T z. \quad (12.10)$$

Простая модификация данной в п. 12.1 статистической интерпретации приводит к выражению

$$\hat{\Lambda}(\hat{x} - x) = \tilde{\Lambda}(\tilde{x} - x) + A^T v. \quad (12.11)$$

Так как  $\mathbf{E}\{\tilde{x} - x\} = 0$  и  $\mathbf{E}\{v\} = 0$ , то и  $\mathbf{E}\{\hat{x} - x\} = 0$ , то есть  $\hat{x}$  есть также несмешённая оценка:  $\mathbf{E}\{\hat{x}\} = x$ . Если ошибка априорной оценки и ошибка измерения взаимно некоррелированы,  $\mathbf{E}\{(\tilde{x} - x)v^T\} = 0$ , то после «возведения в квадрат» обеих частей (12.11) и осреднения получим ковариацию

$$P_{\hat{x}} = \mathbf{E}\{(\hat{x} - x)(\hat{x} - x)^T\} = (\tilde{\Lambda} + A^T A)^{-1} = \hat{\Lambda}^{-1}, \quad (12.12)$$

где через  $\hat{\Lambda}$  обозначена информационная матрица *апостериорной* оценки  $\hat{x}$ ,  $\hat{\Lambda} = \tilde{\Lambda} + A^T A$ .

**Замечание 12.1.** Матрица  $\tilde{\Lambda}$  не обязана быть невырожденной, хотя в (12.7) это формально предполагалось. В действительности, априорное значение некоторых (или всех) компонент вектора  $x$  может быть исчезающе мало, так что соответствующие строки и столбцы в информационной матрице  $\tilde{\Lambda}$  заполняются исчезающе малыми числами или даже нулями. При этом соотношение (12.7) сохраняет силу в пределе, в том смысле, что в ковариационной матрице  $\tilde{P}$  соответствующие диагональные элементы стремятся к  $+\infty$ , в то время как другие остаются ограниченными. Заметьте, что (12.10) как условие минимума функционала (12.8) получается при произвольной неотрицательно определённой матрице  $\tilde{\Lambda}$ . Если  $\tilde{\Lambda} = 0$ , то (12.10) сводится к (11.3) и (12.12) сводится к (12.6). Таким образом, информационная и ковариационная матрицы взаимно обратны не только формально в силу определения (12.7), но и по существу как меры достоверности и, соответственно, недостоверности априорной оценки  $\tilde{x}$  вектора  $x$ .

## 12.3 Включение предшествующего МНК-решения

Расширенная система (12.9) показала, что априорные статистические сведения о векторе  $x$ , поступающие в виде несмешённой оценки  $\tilde{x}$  и её ковариации  $\tilde{P}$ , могут быть интерпретированы как воображаемые добавочные результаты  $\tilde{z} = \tilde{R}\tilde{x}$  некоего эксперимента. Это наводит на мысль, что и чисто алгебраическую задачу отыскания МНК-решения системы уравнений можно решать

последовательно: предварительно найти  $\tilde{x}$  как МНК-решение части системы, а затем включить это  $\tilde{x}$  в полную систему, чтобы найти её МНК-решение  $\hat{x}$ . Такое разделение системы на две части — априорную и текущую — можно назвать её «расщеплением». Пусть система уравнений произвольно расщеплена на эти две подсистемы:

$$\begin{bmatrix} f \\ z \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ A \end{bmatrix} x + \begin{bmatrix} w \\ v \end{bmatrix}. \quad (12.13)$$

МНК-решение  $\hat{x}$  этой полной системы, доставляющее минимум функционалу  $J_1(x) = w^T w + v^T v$ , есть решение нормальных уранений

$$\left[ \begin{array}{c|c} \tilde{R}^T & A^T \end{array} \right] \begin{bmatrix} \tilde{R} \\ A \end{bmatrix} \hat{x} = \left[ \begin{array}{c|c} \tilde{R}^T & A^T \end{array} \right] \begin{bmatrix} f \\ z \end{bmatrix}. \quad (12.14)$$

Допустим, найдено МНК-решение  $\tilde{x}$  для подсистемы  $f = \tilde{R}x + w$  из критерия минимума функционала  $J(x) = w^T w$ . Как отмечено в (11.5)–(11.6), оно удовлетворяет двум условиям:

$$\tilde{R}\tilde{x} = \tilde{z}, \quad \tilde{z} \in \mathcal{R}(\tilde{R}), \quad f - \tilde{z} \perp \mathcal{R}(\tilde{R}),$$

$$\tilde{x} \in \mathcal{R}(\tilde{R}^T).$$

Разностный вектор  $r = f - \tilde{z}$  ортогонален пространству столбцов  $\mathcal{R}(\tilde{R})$  матрицы  $\tilde{R}$  и, следовательно, лежит в левом нуль-пространстве  $\mathcal{N}(\tilde{R}^T)$ , определяемом как все векторы  $y$ , удовлетворяющие уравнению  $\tilde{R}^T y = 0$ . Поэтому

$$\tilde{R}^T f = \tilde{R}^T (\tilde{z} + r) = \tilde{R}^T \tilde{z} + \tilde{R}^T r = \tilde{R}^T \tilde{z}.$$

Следовательно, уравнения (12.14) совпадают с уравнениями

$$\left[ \begin{array}{c|c} \tilde{R}^T & A^T \end{array} \right] \begin{bmatrix} \tilde{R} \\ A \end{bmatrix} \hat{x} = \left[ \begin{array}{c|c} \tilde{R}^T & A^T \end{array} \right] \begin{bmatrix} \tilde{z} \\ z \end{bmatrix},$$

которые, в свою очередь совпадают с уранениями (12.10), так как  $\tilde{R}^T \tilde{R} = \tilde{\Lambda}$ . Тем самым доказано, что МНК-решение  $\hat{x}$  данной системы (12.13) совпадает с МНК-решением системы (12.9), отличающейся от (12.13) тем, что в неё вместо  $f$  включен вектор  $\tilde{z}$ , равный проекции  $f$  на  $\mathcal{R}(\tilde{R})$ ,  $\tilde{z} = \tilde{R}\tilde{x}$ , где  $\tilde{x}$  — МНК-решение левой подсистемы в (12.13).

## 12.4 Рекурсия МНК в стандартной информационной форме

Интерпретация априорных статистических данных как дополнительных наблюдений, или, что равносильно, учёт имеющегося МНК-решения подсистемы после добавления в систему новой порции уравнений, является краеугольным камнем рекурсии для МНК. Это дает возможность обрабатывать экспериментальные данные по мере их поступления, то есть решать задачу о наименьших квадратах по мере поступления новых уравнений. Это равносильно также тому, что исходную большую совокупность экспериментальных данных можно «расщеплять» произвольно на порции и последовательно включать их в обработку. Результат такой последовательной обработки, как доказано выше (подразд. 12.3), будет равносителен результату обработки всей совокупности экспериментальных данных целиком.

Результаты при статистической интерпретации рекурсивны потому, что текущие величины — оценка  $\hat{x}$  и ковариация  $P_{\hat{x}}$  — становятся априорными и комбинируются с новыми данными, чтобы образовать обновлённые оценку и ковариацию. При этом существенно, что результаты (12.10) и (12.12) не зависят (теоретически) от того, какой квадратный корень  $\tilde{R}$  в разложении  $\tilde{\Lambda} = \tilde{R}^T \tilde{R}$  использован. Эта свобода позволяет выбирать  $\tilde{R}$  из соображений большей вычислительной точности. Кроме того, если лишь окончательная оценка (МНК-решение) необходима, то лучше не находить промежуточных оценок, а просто накапливать информационную матрицу  $\sum A_j^T A_j$  и сумму  $\sum A_j^T z_j$ , и лишь в нужный момент (например, в самом конце) вычислить решение.

Информационную форму последовательного МНК запишем, вводя матрицу  $(\Lambda | d)$ .

I. *Инициализация.* Устанавливают начальные значения  $x_0, \Lambda_0$ :

$$d_0 = \Lambda_0 x_0, \quad (\Lambda | d) := (\Lambda_0 | d_0).$$

Эти начальные значения берут из априорных данных:  $x_0 = \tilde{x}, \Lambda_0 = \tilde{\Lambda}$ .

II. *Обработка наблюдений.* Вводят очередную «порцию» наблюдений  $z = Ax + v$ :

$$(\Lambda | d) := (\Lambda | d) + A^T (A | z). \quad (12.15)$$

В общем случае ненормализованных статистических данных  $z$  вместо (12.15) используют алгоритм:

$$(\Lambda | d) := (\Lambda | d) + A^T R^{-1} (A | z). \quad (12.16)$$

III. *Выдача результата.* После последовательной обработки всех порций наблюдений или в нужный момент, когда  $\Lambda^{-1}$  существует, вычисляют

$$\hat{x} = \Lambda^{-1}d, \quad P_{\hat{x}} = \Lambda^{-1}.$$

## 12.5 Рекурсия МНК в стандартной ковариационной форме

Пусть априорная и апостериорная оценки,  $\tilde{x}$  и  $\hat{x}$ , характеризуются невырожденными информационными матрицами  $\tilde{\Lambda}$  и  $\hat{\Lambda}$ , соответственно. Тогда существуют обратные к ним ковариационные матрицы, (12.7) и (12.12). Разрешим (12.10) относительно  $\hat{x}$ :

$$\hat{x} = (\tilde{\Lambda} + A^T A)^{-1} \tilde{\Lambda} \tilde{x} + (\tilde{\Lambda} + A^T A)^{-1} A^T z.$$

Обозначим

$$L = (\tilde{\Lambda} + A^T A)^{-1} \tilde{\Lambda}, \quad K = (\tilde{\Lambda} + A^T A)^{-1} A^T \quad (12.17)$$

и преобразуем:

$$\hat{x} = L \tilde{x} + K z - K A \tilde{x} + K A \tilde{x} = \tilde{x} + K(z - A \tilde{x}),$$

так как  $L + KA = I_n$ . Для определения  $K$  воспользуемся в (12.17) следующей важной леммой.

### Лемма 12.1.

$$(\Lambda_1 - \Lambda_{12}\Lambda_2^{-1}\Lambda_{21})^{-1} = \Lambda_1^{-1} + \Lambda_1^{-1}\Lambda_{12}(\Lambda_2 - \Lambda_{21}\Lambda_1^{-1}\Lambda_{12})^{-1}\Lambda_{21}\Lambda_1^{-1}, \quad (12.18)$$

где предполагается, что все матрицы согласованы по размерам и требуемые обращения матриц существуют.

**Упражнение 12.1.** Докажите лемму 12.1, рассматривая блочные матрицы

$$\begin{bmatrix} \Lambda_1 & \Lambda_{12} \\ \Lambda_{21} & \Lambda_2 \end{bmatrix}^{-1} = \begin{bmatrix} P_1 & P_{12} \\ P_{21} & P_2 \end{bmatrix},$$

и выписывая поблочно равенства  $\Lambda P = I$  и  $P \Lambda = I$ .

Применим лемму 12.1 при  $\Lambda_1 = \tilde{\Lambda}$ ,  $\Lambda_{12} = A^T$ ,  $\Lambda_{21} = \Lambda_{12}^T = A$ ,  $\Lambda_2^{-1} = -I$ . Получим

$$(\tilde{\Lambda} + A^T A)^{-1} = \tilde{\Lambda}^{-1} - \tilde{\Lambda}^{-1} A^T (A \tilde{\Lambda}^{-1} A^T + I)^{-1} A \tilde{\Lambda}^{-1}.$$

Обозначим:  $\tilde{P} = \tilde{\Lambda}^{-1}$ ,  $\hat{P} = \hat{\Lambda}^{-1}$ . Имеем из подразд. 12.3 выражение (12.15):  $\hat{\Lambda} = \tilde{\Lambda} + A^T A$ . Следовательно,  $\hat{P} = \tilde{P} - \tilde{P} A^T (A \tilde{P} A^T + I)^{-1} A \tilde{P}$ . Так как  $K = \hat{P} A^T$ , то

$$\begin{aligned} K &= \tilde{P} A^T - \tilde{P} A^T (A \tilde{P} A^T + I)^{-1} A \tilde{P} A^T = \\ &= \tilde{P} A^T (A \tilde{P} A^T + I)^{-1} [A \tilde{P} A^T + I - A \tilde{P} A^T] = \tilde{P} A^T (A \tilde{P} A^T + I)^{-1}. \end{aligned}$$

Таким образом, при статистической интерпретации (см. подразд. 12.2) получаем возможность уточнять априорную несмещённую оценку  $\tilde{x}$  и уменьшать её ковариацию  $\tilde{P}$ , благодаря включению в процесс обработки вновь поступивших результатов наблюдений  $z = Ax + v$  и применяя к ним следующий алгоритм, известный как *стандартный алгоритм Калмана* (рис. 12.2).

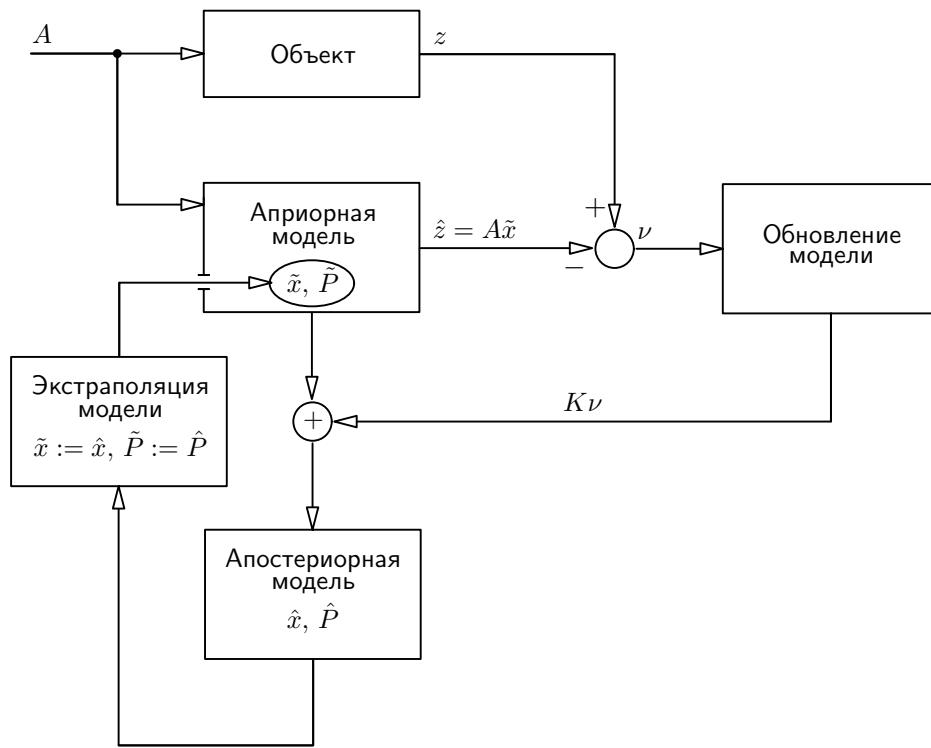


Рис. 12.2. Рекурсия МНК в стандартной ковариационной форме (схема Калмана). Этап 1 – обновление модели по наблюдениям. Этап 2 – экстраполяция модели между наблюдениями. Априорная модель даёт предсказание  $\hat{z}$  для отклика  $z$  объекта. Разность  $\nu$  имеет смысл обновляющего процесса. Весовая матрица  $K$ , умноженная на  $\nu$ , обеспечивает обновление априорной модели по наблюдению  $z$ , т. е. переход к апостериорной модели

I. *Инициализация*. Начальные значения  $x_0$ ,  $P_0$ :

$$\tilde{x} := x_0, \quad \tilde{P} := P_0. \quad (12.19)$$

II. *Обработка наблюдений.* Очередная порция наблюдений  $z = Ax + v$ :

$$\begin{aligned} K &= \tilde{P}A^T(A\tilde{P}A^T + I)^{-1}, \\ \hat{P} &= \tilde{P} - KA\tilde{P}, \\ \hat{x} &= \tilde{x} + K(z - A\tilde{x}). \end{aligned} \quad (12.20)$$

III. *Экстраполяция.* Распространение оценки  $\hat{x}$  и её ковариации  $\hat{P}$  между наблюдениями, т. е. к моменту повторения этапа II со следующей порцией наблюдений:

$$\tilde{P} := \hat{P}, \quad \tilde{x} := \hat{x}. \quad (12.21)$$

**Замечание 12.2.** Первая и вторая формулы в (12.21) выражают правило экстраполяции только для статической МНК-задачи оценивания, т. е. для случая, когда оцениваемый вектор  $x$  не изменяется в процессе наблюдения:  $x = \text{const}$ .

Равным образом данный алгоритм пригоден и без статистической интерпретации (см. подразд. 12.3), когда алгебраическая задача отыскания МНК-решения переопределённой системы решается последовательно. Такое решение может стартовать с условно «пустой» системы уравнений. Практически, это должно отвечать условию  $\tilde{\Lambda} = 0$ , которое легко реализовать в информационной форме (подразд. 12.3). В ковариационной форме данное условие можно реализовать лишь приближенно, например, полагая  $P_0 = \varepsilon^{-2}I$ , где  $\varepsilon \rightarrow 0$ . При таком выборе величина  $x_0$  практически не имеет влияния на дальнейший процесс, так что она может быть взята равной нулевому вектору,  $x_0 = 0$ . После такой инициализации уравнения исходной переопределённой системы могут вводиться в этап обработки измерений последовательными порциями, и в порциях может содержаться любое, не обязательно одно и то же, число уравнений, выражаемое в числе строк матрицы  $A$ .

Как следует из подразд. 12.3, от указанного числа МНК-решение всей алгебраической системы уравнений не зависит. В связи с этим, с вычислительной точки зрения удобным оказывается добавление в очередной порции лишь по одному уравнению. Тогда матрица  $A$  содержит всего одну строку, которую теперь обозначим как транспонированный вектор-столбец  $a$ ,  $a^T = (a_1, a_2, \dots, a_n)$ . В этом случае

$$z = a^T x + v, \quad (12.22)$$

и обработка наблюдений (12.20) принимает особенно простой вид:

$$\alpha = a^T \tilde{P} a + 1, \quad K = \tilde{P} a / \alpha, \quad \hat{P} = \tilde{P} - K a^T \tilde{P}, \quad \hat{x} = \tilde{x} + K(z - a^T \tilde{x}). \quad (12.23)$$

Это алгоритм скалярной (последовательной) обработки. В нём вместо умножения на обратную матрицу  $(A\tilde{P}A^T + I)^{-1}$  применяется деление на скалярную величину  $\alpha$ .

**Упражнение 12.2.** С применением леммы 12.1 к выражению (12.16) докажите, что в общем случае ненормализованных экспериментальных данных (11.1) при их статистической интерпретации с ковариацией ошибок  $v$ , равной  $R$ , матрица Калмана  $K$  в алгоритме (12.20) определяется выражением

$$K = \tilde{P}A^T(A\tilde{P}A^T + R)^{-1}. \quad (12.24)$$

**Упражнение 12.3.** Статистическая интерпретация алгоритма (12.23) дана в подразд. 12.1 для случая, когда экспериментальные данные (11.1) нормализованы, то есть характеризуются математическими ожиданиями (12.3). Это выражается добавлением «+1» в выражении для  $\alpha$ , (12.23), причём эта «+1» есть не что иное как  $\mathbf{E}\{vv^T\} = 1$  для ошибки  $v$  в (12.22). Докажите, что в более общем случае, когда матрица  $R$  в (12.24) является диагональной,

$$R = \text{diag}(r_1, r_2, \dots, r_m), \quad (12.25)$$

и только в этом случае матричный алгоритм (12.20) с матрицей  $K$  из (12.24) эквивалентен  $t$ -кратному повторению скалярного алгоритма вида (12.23) с  $\alpha = a^T\tilde{P}a + r$ , где  $a^T$  –  $i$ -я строка матрицы  $A$ ,  $r = r_i$  и  $z$  –  $i$ -й элемент вектора  $z$ , (11.1), при  $i = 1, 2, \dots, t$ .

Таким образом, скаляризованный алгоритм Калмана (12.23) имеет следующее общее представление:

$$\alpha = a^T\tilde{P}a + r, \quad K = \tilde{P}a/\alpha, \quad \hat{P} = \tilde{P} - Ka^T\tilde{P}, \quad \hat{x} = \tilde{x} + K(z - a^T\tilde{x}), \quad (12.26)$$

если наблюдения (11.1) в их статистической интерпретации составлены из  $t$  отдельных, независимых друг от друга, скалярных данных вида (12.22), каждое с ковариацией  $r = r_i$ ,  $i = 1, 2, \dots, t$ .

Ещё раз отметим, что в применении к решению переопределённой системы алгебраических уравнений в (12.26) следует считать  $r = 1$ , то есть использовать (12.23).

**Замечание 12.3.** Условие (12.25) не является ни в коей мере ограничительным для использования (12.26). Используя разложение Холесского без квадратных корней (см. разд. 4.3), любую  $R > 0$  можно представить в виде  $R = UDU^T$  или  $R = LDL^T$  и затем перейти к измерениям  $\bar{z} = U^{-1}z$  или  $\bar{z} = L^{-1}z$ , чтобы диагонализировать матрицу ковариаций ошибок наблюдений.

**Упражнение 12.4.** Докажите, что в алгоритме (12.20) с определением  $K$  по выражению (12.24) вычисление  $\hat{P}$  может быть представлено в так называемой симметричной форме Джозефа:

$$\hat{P} = (I - KA)\tilde{P}(I - KA)^T + KRK^T,$$

которую иначе называют *стабилизированным алгоритмом Калмана*, так как она предотвращает возможную потерю положительной определённости матрицы  $\hat{P}$ , присущую стандартному алгоритму (12.20) с  $\hat{P} = \tilde{P} - KAP$ . При скалярной обработке наблюдений в алгоритме (12.26) это выражение для  $\hat{P}$ , соответственно, заменяется на *скаляризованный алгоритм Джозефа*

$$\hat{P} = (I - K\alpha^T)\tilde{P}(I - \alpha K^T) + rKK^T.$$

## 12.6 Ковариационный алгоритм Поттера для МНК

Вместо матриц  $\tilde{P}$  и  $\hat{P}$ , по своей природе положительно определённых, далее оперируем с квадратными корнями из них, соответственно,  $\tilde{S}$  и  $\hat{S}$ , отвечающими равенствам  $\tilde{S}\tilde{S}^T$  и  $\hat{S}\hat{S}^T$ . Перепишем выражение для  $\hat{P}$  в (12.26) в виде

$$\hat{S}\hat{S}^T = \tilde{S}(I_n - ff^T/\alpha)\tilde{S}^T, \quad f = \tilde{S}^Ta, \quad \alpha = r + f^Tf,$$

где  $n$  – размерность векторов  $\hat{x}$ ,  $\tilde{x}$ , и потребуем так выбрать число  $\beta$ , чтобы обеспечить справедливость следующего разложения:

$$I_n - ff^T/\alpha = (I_n - \beta ff^T)(I_n - \beta ff^T).$$

Отсюда для  $\beta$  получаем квадратное уравнение и из двух его решений выбираем

$$\beta = (1/\alpha)/(1 + \sqrt{r/\alpha}),$$

поскольку выбор знака «+» обеспечивает меньший уровень относительных ошибок при этих вычислениях. Обозначим

$$\gamma = 1/(1 + \sqrt{r/\alpha}),$$

тогда  $\beta = \gamma/\alpha$ . В итоге вместо (12.26) получаем следующий ряд формул:

$$\left. \begin{array}{l} f = \tilde{S}^Ta, \\ \alpha = f^Tf + r, \\ \gamma = 1/(1 + \sqrt{r/\alpha}), \\ K = \tilde{S}f/\alpha, \\ \hat{S} = \tilde{S} - \gamma Kf^T, \\ \hat{x} = \tilde{x} + K(z - a^T\tilde{x}), \end{array} \right\} \quad (12.27)$$

который и составляет алгоритм Поттера. Он численно более устойчив, чем стандартный ковариационный алгоритм Калмана (12.26), но ему эквивалентен. В целом, для него характерно следующее.

Вычисление  $\hat{S}$  в (12.27) равносильно счёту с двойной точностью матрицы  $\hat{P}$  в (12.26) при использовании той же разрядности чисел в компьютере, или, иначе, равносильная точность вычислений матрицы  $\hat{P}$  может быть достигнута значительно быстрее. Для матрицы  $\hat{P}$  теперь отсутствует опасность потери положительной определённости, присущая операции вычитания в (12.26), поскольку здесь вычисляют  $\hat{S}$ , а  $\hat{P} = \hat{S}\hat{S}^T$  и  $\hat{P} > 0$  при  $\det(\hat{S}) \neq 0$ . Недостатком алгоритма (12.27) является наличие операции извлечения квадратного корня, отдельной для каждого скалярного наблюдения  $z = a^T x + v$ , и возможная потеря специального (желательно, треугольного) вида матрицы  $\hat{S}$  в общем случае. Действительно, для экономии памяти и объёма вычислений обычно стремятся иметь матрицы  $\hat{S}$  и  $\tilde{S}$  в виде треугольных матриц (обе – нижние треугольные или обе – верхние треугольные), что соответствует разложениям Холлесского:  $\hat{P} = \hat{S}\hat{S}^T$  и  $\tilde{P} = \tilde{S}\tilde{S}^T$ . Однако, если стартовать с матрицы  $\tilde{S}$  треугольного вида, выполняя инициализацию в соответствии с (12.19), то из-за вычитания в (12.27) матрица  $\hat{S}$  в общем случае не остается треугольной. Например, пусть для  $\hat{S}$  и  $\tilde{S}$  выбрана верхняя треугольная форма. Тогда только при  $a = \lambda(1, 0, \dots, 0)^T$ , где  $\lambda$  – скаляр, для  $\hat{S}$  в (12.27) будет сохранена та же верхняя треугольная форма, благодаря чему выполнение этапа экстраполяции, согласно (12.21), сводится к простому присваиванию:  $\tilde{S} := \hat{S}$ . Если же выбранная для  $\tilde{S}$  треугольная форма будет утрачена, то этап экстраполяции матрицы потребует предварительной триангуляризации матрицы  $\hat{S}$ , то есть операции  $\tilde{S} := \text{triang}(\hat{S})$ . Триангуляризация  $\text{triang}(\cdot)$  должна быть выполнена ортогональным преобразованием матрицы  $(\cdot)$ , например, преобразованиями Хаусхолдера, Гивенса или же Грама-Шмидта.

## 12.7 Задание на лабораторный проект № 9

### Введение

В данном лабораторном проекте мы предлагаем к изучению современные методы построения численно устойчивых и экономичных по затратам ресурсов ЭВМ алгоритмов метода наименьших квадратов (МНК), решающих актуальную задачу последовательного обновления оценок по измерениям. Как уже отмечалось, эта задача возникает во многих приложениях, например, при оценке состояния (элементов движения) объекта по последовательно поступающим

данным наблюдения, при подгонке параметров модели под результаты продолжительных экспериментов, проводимых в физике, астрономии, экономике, бизнесе, социологии, психологии и в других областях, где выявление регрессий, анализ и прогнозирование тенденций опирается на включение в обработку данных наблюдения по мере их поступления, для того, чтобы постепенно идентифицировать модель объекта (процесса) или уточнять параметры модели.

## 1. Задание

**А.** Спроектировать, отладить и продемонстрировать в действии программу решения несовместной системы уравнений  $Ax \approx b$ ,  $A = A(m, n)$ ,  $m > n$ ,  $\text{rank } A = n$ , в смысле наименьших квадратов в соответствии с вашим вариантом последовательного алгоритма (см. ниже подразд. 12.9).

**Б.** Оценить результаты решения по трем показателям:

- (1) погрешность (абсолютная и относительная) решения;
- (2) затраты основной памяти компьютера на хранение данных, необходимых только для заданного алгоритма;
- (3) теоретическое и реальное число основных операций компьютера для выполнения заданного алгоритма.

Эти показатели определить в зависимости от следующих параметров задачи:

- (1) размерность задачи, т. е. число неизвестных  $n$ ;
- (2) степень переопределённости задачи, т. е. число  $p$ , указывающее, во сколько раз число уравнений  $m$  больше числа неизвестных,  $m = pn$ ;
- (3) степень несовместности системы, т. е. вещественное положительное число  $c$ , показывающее среднеквадратическое значение элементов случайного разностного вектора  $d = b - A\bar{x}$ , где  $\bar{x}$  – нормальное псевдорешение системы  $Ax \approx b$ , относительно среднего (единичного) значения;
- (4) способ генерации матрицы  $A$ .

**В.** Провести вычислительный эксперимент, используя в нём:

- (1) десять значений  $n$ ,  $n = 1, 2, \dots, 10$ ;
- (2) три значения  $p$ ,  $p = 10, 100$  и  $1000$ ;
- (3) три значения  $c$ ,  $c = 1/10, 1$  и  $10$ ;

(4) четыре способа генерации матрицы  $A$  (см. п. 2 в подразд. 12.7).

Результаты эксперимента вывести на экран в виде следующей таблицы:

### Вычислительный эксперимент:

$p=(\text{значение}), \quad c=(\text{значение}), \quad A=(\text{способ})$				
$n$	Погрешность		Память КБайт	Число операций теоретическое   реальное
	абсолютная	относительная		

При подсчёте числа операций учитывать: сложение, умножение, деление и извлечение квадратного корня. К отчёту о работе приложить расчётные формулы числа операций отдельно по этим видам операций и их сумму. В таблицу выводить только суммарное число операций.

**Г.** Выполнить отладку программы и продемонстрировать результаты на решении следующей тестовой задачи [14]:

$$Ax \approx b, \quad A = A(m, 2) = \begin{bmatrix} a_{i1} & : & a_{i2} \end{bmatrix}, \quad i = 1, 2, \dots, m;$$

$$m = 4, 8, 12, 16, 20, 24, 28, 32, 36, 40;$$

$$a_{i1} = w_i = \sin(2\pi i/m), \quad a_{i2} = w_{i-1};$$

$$b^T = (b_1, \dots, b_m), \quad b_i = 2 \cos(2\pi i/m).$$

Известно (с. 393), что решением этой задачи является вектор  $\bar{x}^T = (\bar{x}_1, \bar{x}_2)$ ,

$$\bar{x}_1 = 2 \operatorname{ctg}(2\pi/m), \quad \bar{x}_2 = -2 \operatorname{cosec}(2\pi/m). \quad (12.28)$$

Для демонстрации процесса отладки вывести на экран ещё одну таблицу:

### Отладка программы:

$m$	Погрешность	
	абсолютная	относительная

**Д.** Во всех случаях для оценки абсолютной погрешности использовать норму вектора  $e = x - \bar{x}$  вида

$$\|e\|_\infty = \max_i |e_i|,$$

где  $x$  – вычисленное решение, а относительную погрешность определять по формуле

$$\delta = \|e\|_\infty / \|\bar{x}\|_\infty,$$

в которой  $\bar{x}$  – точное МНК-решение (нормальное псевдорешение задачи). В отладочной (тестовой) задаче по п. Г решением является вектор (12.28), а в задачах вычислительного эксперимента по п. В – вектор  $\bar{x} = (1, 2, \dots, n)$ .

## 2. Генерация задач для вычислительного эксперимента

Как отмечено, для этих задач точное МНК-решение следует задать в виде вектора  $\bar{x} = (1, 2, \dots, n)$ . Затем следует сгенерировать матрицу  $A$  (см. ниже) и образовать вектор  $\hat{b} = A\bar{x}$ . К нему нужно добавить случайный вектор  $d = c\xi$ , где  $c$  – число (см. выше п. 1), а  $\xi \sim \mathcal{N}(0, 1)$  – вектор случайных чисел (от подпрограммы псевдослучайных чисел), взятых из стандартного нормального распределения с нулевым средним значением и единичной дисперсией. В результате получаем вектор  $b = A\bar{x} + d$  для системы уравнений  $Ax \approx b$ .

Для генерации матрицы  $A$  необходимо предусмотреть один из четырёх способов:

**Способ 1** – матрица  $A = A(m, n)$  заполняется случайными числами из равномерного распределения в диапазоне  $[-100, +100]$ . Условно запишем это так:

$$A = [\text{Random}(m \times n)].$$

**Способ 2** – верхняя часть матрицы  $A$ , а именно, её первые  $n$  строк, заполняются по способу 1, а остальные строки образуют подматрицу, в которой только первые  $q$  столбцов, где  $q = [n/2]$  – ближайшее снизу целое число к  $n/2$ , заполняются как в способе 1, а все остальные столбцы – нулевые. Таким образом, матрица имеет вид:

$$A = \begin{array}{|c|} \hline \text{Random}(n \times n) \\ \hline \text{Random} & 0 \\ \hline \end{array}$$

**Способ 3** – первая часть матрицы  $A$  должна формироваться как в способе 2, а остальная часть образуется располагающимися последовательно вниз блоками из единичных матриц  $I$  размера  $n \times n$ :

$$A = \begin{array}{|c|} \hline \text{Random}(n \times n) \\ \hline I(n \times n) \\ \cdots \\ I(n \times n) \\ \hline \end{array}$$

**Способ 4** – верхняя часть матрицы  $A$  должна формироваться как в способе 2, остальная же часть строится подобно способу 2, но ненулевые  $q$  столбцов нижней подматрицы заполняются не случайными числами, а располагающимися последовательно вниз блоками из единичных матриц  $I$  размера  $(q \times q)$ :

$$A = \begin{array}{|c|c|} \hline & \text{Random}(n \times n) \\ \hline I(q \times q) & \\ \dots & 0 \\ I(q \times q) & \\ \hline \end{array}$$

**Замечание 12.4.** Не нужно генерировать всю матрицу  $A$ , так же как и весь вектор  $b$  и весь вектор  $d$ , единовременно. Матрицу  $A$  нужно генерировать построчно, а векторы  $b$  и  $d$  – поэлементно. Например, если  $a^T = (a_1, a_2, \dots, a_n)$  есть текущая строка матрицы  $A$ , то  $z = a^T \bar{x} + v$ , где  $z$  – текущий элемент вектора  $b$ ,  $v$  – текущий элемент вектора  $d$ ,  $v = c\xi$ ,  $\xi \sim \mathcal{N}(0, 1)$  – текущее случайное число из стандартного нормального распределения. Таким образом, последовательно генерируемые данные  $(a^T, z)$  нужно вводить в алгоритм решения, а также использовать в нём значение  $r = c^2$ , имеющее смысл дисперсии ошибки измерения вектора  $\hat{b} = A\bar{x}$ , исключительно последовательно.

## 12.8 Варианты задания на лабораторный проект № 9

Общее число вариантов составляет 16 (учитывая подварианты – различия в методах ортогонализации в некоторых из вариантов).

**Замечание 12.5.** Для всех ковариационных алгоритмов (варианты 1–3) в качестве начальных значений можно взять:  $x_0 = 0$ ,  $P_0 = (1/\varepsilon^2)I$ ,  $\varepsilon \rightarrow 0$ .

**Вариант 1.** Стандартный ковариационный алгоритм (Калмана). Найдите его в подразд. 12.5.

**Вариант 2.** Стабилизированный ковариационный алгоритм (Джозефа). Найдите его в подразд. 12.5:

(ii) Обработка наблюдений (очередные данные  $z = a^T \bar{x} + v$ ):

$$\alpha = a^T \tilde{P} a + r, \quad K = \tilde{P} a / \alpha,$$

$$\hat{P} = (I - K a^T) \tilde{P} (I - a K^T) + r K K^T. \quad (12.29)$$

**Замечание 12.6.** Вычислительные затраты существенно зависят от способа программирования выражений. Например, выражение (12.29) для  $\hat{P}$

может быть запрограммировано в следующей последовательности:

$$\begin{aligned} W_1 &= I - Ka^T, && (n \times n)\text{-матрица} \\ W_2 &= W_1 \tilde{P}, && (n \times n)\text{-матрица} \\ \hat{P} &= W_2 W_1^T + r(KK^T) \end{aligned}$$

или, эквивалентно, в виде:

$$\begin{aligned} v_1 &= \tilde{P}a, && n\text{-вектор} \\ P_1 &= \tilde{P} - Kv_1^T, && (n \times n)\text{-матрица} \\ v_2 &= P_1 a, && n\text{-вектор} \\ \hat{P} &= (P_1 - v_2 K^T) + (rK)K^T, \end{aligned}$$

и в обоих способах можно экономить вычисления, благодаря симметрии матрицы  $\hat{P}$ . Однако второй способ имеет на порядок меньше вычислений: в первом способе выполняется  $(1, 5n^3 + 3n^2 + n)$  умножений, а во втором только  $(4n^2 + 2n)$  умножений.

**Вариант 3.** Квадратно-корневой ковариационный алгоритм Поттера. Найдите его в подразд. 12.6, в следующем виде.

(i) Инициализация (начальные значения  $x_0$ ,  $P_0$ ):

$$\tilde{x} := x_0, \quad \tilde{S} := P_0^{1/2}.$$

(ii) Обработка наблюдений (очередные данные  $z = a^T \bar{x} + v$ ):

$$\begin{aligned} f &= \tilde{S}^T a, \quad \alpha = f^T f + r, \quad \gamma = 1/(1 + \sqrt{r/\alpha}), \\ K &= \tilde{S} f / \alpha, \quad \hat{S} = \tilde{S} - \gamma K f^T, \\ \hat{x} &= \tilde{x} + K(z - a^T \tilde{x}). \end{aligned}$$

(iii) Экстраполяция (между повторениями этапа (ii)):

$$\tilde{S} := \hat{S}, \quad \tilde{x} := \hat{x}.$$

**Замечание 12.7.** Вариант 3, в котором вместо  $\tilde{S} := \hat{S}$  предусмотрена процедура триангуляризации  $\tilde{S} := \text{triang } \hat{S}$ , даёт следующие версии:  
– Версия 3.1: обе матрицы,  $\tilde{S}$  и  $\hat{S}$ , – нижние треугольные ( $S \equiv L$ ), или

– Версия 3.2: обе матрицы,  $\tilde{S}$  и  $\hat{S}$ , – верхние треугольные ( $S \equiv U$ ).

Именно для этого этап (iii) должен содержать, вместо  $\tilde{S} := \hat{S}$ , процедуру триангуляризации  $\tilde{S} := \text{triang } \hat{S}$ , матрицы  $\hat{S}$ . Возможны четыре алгоритма этой процедуры: (1) отражения Хаусхолдера, (2) вращения Гивенса, (3) классическая Грама–Шмидта ортогонализация и (4) модифицированная Грама–Шмидта ортогонализация (см. лабораторный проект № 6). Соответственно этому, всего имеем 8 подвариантов для указанного варианта 3, сведённых в следующую таблицу:

$\text{triang}$	$S \equiv L$	$S \equiv U$
Хаусхолдер	3.1.1	3.2.1
Гивенс	3.1.2	3.2.2
ГШО	3.1.3	3.2.3
МГШО	3.1.4	3.2.4

**Вариант 4.** Стандартный информационный алгоритм (см. стр. 401).

(i) Инициализация (начальные значения  $x_0, \Lambda_0$ ):

$$d_0 = \Lambda_0 x_0, \quad \left( \begin{array}{c:c} \Lambda & d \end{array} \right) := \left( \begin{array}{c:c} \Lambda_0 & d_0 \end{array} \right).$$

(ii) Обработка наблюдений (очередные данные  $z = a^T \bar{x} + v$ ):

$$\left( \begin{array}{c:c} \Lambda & d \end{array} \right) := \left( \begin{array}{c:c} \Lambda & d \end{array} \right) + a \left( \begin{array}{c:c} a^T & z \end{array} \right) / r.$$

(iii) Выдача результата:  $\hat{x} = \Lambda^{-1} d$ .

В качестве начальных значений рекомендуется взять  $x_0 = 0, \Lambda_0 = 0$ .

**Вариант 5.** Квадратно-корневой информационный алгоритм. См. подразд. 5.3, стр. 169, формулу (5.7), здесь – её рекуррентная версия (12.30).

(i) Инициализация (начальные значения  $\tilde{R}_0, x_0$ ):

$$\tilde{z}_0 = \tilde{R}_0 x_0; \quad \left[ \begin{array}{c|c} \tilde{R}_0 & \tilde{z}_0 \end{array} \right] = \left[ \begin{array}{c|c} \tilde{R}_0 & \tilde{z}_0 \end{array} \right].$$

(ii) Обработка наблюдений (очередные скалярные данные  $z = a^T \bar{x} + v$ ):

$$\left[ \begin{array}{cc} \hat{R}_j & \hat{z}_j \\ 0 & e \end{array} \right] = T_j \left[ \begin{array}{cc} \hat{R}_{j-1} & \hat{z}_{j-1} \\ a^T & z \end{array} \right], \quad j = 1, 2, \dots, m, \quad (12.30)$$

где  $T_j$  – ортогональная матрица, которая выбирается так, чтобы привести к верхнетреугольному виду расширенную матрицу  $\begin{bmatrix} \hat{R}_{j-1} \\ a^T \end{bmatrix}$ ,  $j$  – номер очередного измерения, все матрицы  $R$  здесь имеют размер  $(n \times n)$ , при этом все  $R_j$ ,  $j \geq 1$ , – верхние треугольные.

(iii) Выдача результата:  $\hat{x} = \hat{R}_j^{-1} \hat{z}_j$ .

Начальные значения рекомендуется взять в виде  $x_0 = 0$ ,  $\tilde{R}_0 = 0$ . В качестве преобразований  $T_j$  возможны четыре процедуры, указанные в описании варианта 3. Таким образом, всего имеем 4 разновидности данного варианта:

$T_j$	вариант
Хаусхолдер	15.1
Гивенс	15.2
ГШО	15.3
МГШО	15.4

## 12.9 Заключение по разделу 12

В данном разделе рассмотрен современный метод наименьших квадратов для решения несовместных (переопределённых) систем линейных алгебраических уравнений. Он назван «современным», потому что появился относительно недавно (известен с 1963 года). Он сводится к решению так называемой нормальной системы уравнений, при этом все уравнения этой системы решаются не одновременно, а последовательно.

Этот метод имеет такую же алгоритмическую форму, как знаменитый, широко применяемый фильтр Калмана. Это совпадение алгоритмов относится лишь к этапу обработки измерений в фильтре: этап экстраполяции оценок здесь отсутствует, так как оценивается «статический» вектор решения переопределённой алгебраической системы. Изучение этого метода крайне важно с прикладной точки зрения, но важно и с точки зрения вычислительной математики, поскольку позволяет преодолевать проблему плохой обусловленности системы.

В проекте № 9 предлагается выполнить программирование нескольких (трёх первоначальных) последовательных алгоритмов метода наименьших квадратов и исследовать их. Этот проект является специальным для освоения студентами дисциплин «Вычислительная математика» или «Численные методы».

Более широко алгоритмы этого раздела 12 представлены в базовом учебном пособии [6], где для них дана полная статистическая интерпретация.

# Заключение

Данное учебное пособие разработано с учётом современных тенденций проектно-ориентированного преподавания и обучения. В рамках ПОО студент не просто программирует метод, а создаёт академический программный продукт (АПП), который должен приближаться по качеству к профессиональным программным продуктам.

Структурно данное пособие организовано в виде четырёх частей, в которые помещены двенадцать разделов.

Часть I написана на материале, который предоставили доцент В. В. Угаров и аспирант А. И. Афанасова, – ими разработаны методика оценивания качества АПП (В. В. Угаров, с. 39–43) и программа вычисления качественных характеристик совокупности АПП на основе метрик Холстеда (А. И. Афанасова, с. 43–46). Эти методика и инструменты используются в Ульяновске, начиная с 1999 года, когда основным изучаемым языком программирования был Pascal и среда Delphi. Теперь, когда студенты изучают и практически используют языки C++, C# и MATLAB, инструменты этой методики (программы–анализаторы качества АПП) обновляются. В них осреднённая оценка качества АПП выводится с применением статистических методов по большому количеству АПП, сохраняемых в базе данных.

Доцент В. В. Воронина включила в часть I методические рекомендации по использованию языка программирования высокого уровня C# при выполнении проектных заданий студентами. Ею собраны базовые сведения по C# и написаны полезные фрагменты кодов (В. В. Воронина, с. 52–83).

Часть II, содержащую три проекта, написали профессор И. В. Семушин (теоретический материал, с. 87–114, 135–151, 166–198), доцент Ю. В. Цыганова (методические рекомендации по выполнению проектов, с. 114–134, 151–165, 199–220) и студент И. Н. Куличенко, – ему принадлежит раздел 6, с. 221–314. Раздел 6 предназначен для демонстрации всего рабочего процесса разработки любого из вариантов не только первых трёх проектов, но и других шести, помещённых в следующие части данного пособия. Результат разработки одного из вариантов проекта № 1 доступен на долговременном ресурсе Google.

## Распределение числа вариантов заданий на проектирование

Часть №	Проект №	Число вариантов задания	Указатель: раздел ?? с. ??	Всего вариантов
II	1	26	разд. 3.8, с. 112	94
	2	40	разд. 4.8, с. 151	
	3	28	разд. 5.16, с. 198	
III	4	40	разд. 7.7, с. 333	71
	5	16	разд. 8.5, с. 347	
	6	15	разд. 9.12, с. 366	
IV	7	48	разд. 10.4, с. 386	92
	8	28	разд. 11.5, с. 394	
	9	16	разд. 12.9, с. 414	
II + III + IV	Суммарное число вариантов			257

Части III и IV пособия разработал профессор И. В. Семушин (с. 317–414). Этот материал может быть полезен для углублённого изучения методов и алгоритмов Вычислительной линейной алгебры, включая начальные аспекты технологии разреженных матриц и метода обыкновенных наименьших квадратов. Последнее особенно востребовано для современной реализации решений задач регрессионного анализа, например, в эконометрике, где до сих пор распространено одновременное решение нормальных уравнений. Дальнейший материал по этим важным вопросам может быть найден в книгах [6], [9] и [11].

Фронтально-состязательный подход и ПОО, которые мы реализуем в этом образовательном процессе, нуждаются в большом разнообразии индивидуальных вариантов проекта по одной и той же изучаемой теме. Эта проблема «размножения» вариантов в данном пособии решена. В результате в данном учебном пособии мы имеем по всем девяти тематическим проектам 257 различных вариантов индивидуальных заданий на проектирование алгоритмов вычислительной линейной алгебры (см. таблицу выше) с реализацией проектов на языке программирования высокого уровня C#. Это существенно отличается по количеству проектов, по систематизации методики проектирования и по современности языка от тех учебных проектов, которые описаны в книге [12].

# Список иллюстраций

2.1	Создание проекта . . . . .	56
2.2	Выбор вида проекта (а). Корректный результат (б) . . . . .	57
2.3	Преобразование типов . . . . .	59
2.4	Основная форма (а). Размещение текстовых полей: выбор элемента TextBox в CommonControls (б). Вкладка свойств основной формы (в) . . . . .	66
2.5	Первый этап создания интерфейса . . . . .	67
2.6	Обработка нажатия на кнопку (а). Настройка параметров таблицы (б) . . . . .	68
2.7	Работа приложения . . . . .	71
2.8	Отображение графика . . . . .	75
2.9	Отображение гистограммы . . . . .	76
2.10	Добавление класса . . . . .	78
4.1	Алгоритмы окаймления известной части $LL^T$ -разложения: строчный (слева); алгоритм скалярных произведений (справа) . . . . .	145
4.2	Алгоритмы окаймления неизвестной части $LL^T$ -разложения: алгоритм линейных комбинаций (слева); алгоритм скалярных произведений (справа) . . . . .	145
5.1	Алгебраически эквивалентные задачи, решаемые методом наименьших квадратов значений невязки $v$ или среднего квадрата погрешности $e$ : (а) – оптимальное моделирование неизвестной системы по экспериментальным условиям $A$ и данным $z$ ; (б) – оптимальное оценивание неизвестного вектора по наблюдениям $Ax$ в присутствии случайных помех $v$ с характеристиками $\mathbf{E}\{v\} = 0$ и $\mathbf{E}\{vv^T\} = I$ . . . . .	167
5.2	Геометрия преобразования Хаусхолдера. Задача 1 (прямая): даны векторы $u$ и $y$ , найти вектор $y_r$ , отражённый от гиперплоскости $U_\perp$ . . . . .	171
5.3	Геометрия преобразования Хаусхолдера. Задача 2 (обратная): даны векторы $y$ и $y_r$ , найти вектор $u$ , задающий отражающую гиперплоскость $U_\perp$ ; здесь $y_r = se_1 = [s \mid 0 \cdots 0]^T$ . Докажите, что здесь показан вектор $\frac{1}{2}u$ , а не $u$ (см. Замечание 5.2) .	173
5.4	Представление возможных случаев применения теоремы 5.1 к матрице $A(m, n)$ ; (а) недоопределённая система: $k = m - 1 \leq n$ ; (б) определённая система: $k = n - 1$ , $m = n$ ; (с) переопределённая система: $k = n < m$ ; (д) $k < n < m$ . . . . .	174
5.5	<i>Вверху:</i> Сохранение преобразования $T$ и вычисление вектора $y = Tz$ , $\forall y \in \mathbb{R}^m$ . <i>Внизу:</i> Вычисление матрицы $A^{-1}$ после сохранения преобразования $T$ . . . . .	180
5.6	Геометрия вращений . . . . .	181
5.7	Вычисление матрицы $P_{1,j}$ . . . . .	182
5.8	Преобразование Гивенса: (а) столбцово ориентированная схема вычисления матрицы $PA$ , где $P = P^{(j)}$ при $j = \min(m - 1, n)$ (нижняя матрица слева); (б) вычисление координаты $r$ вектора $(a, b)^T$ , повёрнутого до совмещения с первой осью, а также косинуса и синуса угла поворота и рабочего признака $\zeta$ ; (в) строчно ориентированная схема вычисления матрицы $PA$ (верхняя матрица слева); (г) восстановление косинуса и синуса угла поворота из признака $\zeta$ ; (д) получение вектора $y$ теми преобразованиями $P_{j,i}$ произвольного вектора $z \in \mathbb{R}^m$ , которые сохранены в рабочих признаках $\zeta_{j,i}$ и восстанавливаются из них; (е) вследствие п. (б) векторы 1, 2, 3 и 4 поворачиваются к положительному направлению первой координатной оси, а векторы 5, 6, 7 и 8 – к отрицательному направлению этой оси . . . . .	184
6.1	Вывод матрицы после компиляции процедуры факторизации . . . . .	226
6.2	Вывод ошибки . . . . .	231
6.3	Вывод решения СЛАУ . . . . .	238

6.4	Вывод числа и времени операций . . . . .	238
6.5	Вывод определителя матрицы . . . . .	240
6.6	Создание d11 библиотеки . . . . .	247
6.7	Вывод случайной матрицы . . . . .	253
6.8	Вывод матрицы первого типа . . . . .	253
6.9	Вывод матрицы второго типа . . . . .	255
6.10	Вывод матрицы четвёртого типа. Иной вариант: вместо операции умножения "*" в седьмой строке листинга 6.35 здесь применена операция деления "/". Рекомендуем использовать основной вариант листинга 6.35 . . . . .	256
6.11	Вывод матрицы пятого типа. Иной вариант: вместо операции умножения "*" в седьмой строке листинга 6.37 здесь применена операция деления "/". Рекомендуем использовать основной вариант листинга 6.37. Подобное замечание см. для рис. 6.10 . . . . .	258
6.12	Вывод матрицы шестого типа . . . . .	261
6.13	Вывод матрицы седьмого типа . . . . .	262
6.14	Вывод матрицы восьмого типа . . . . .	263
6.15	Вывод матрицы девятого типа . . . . .	265
6.16	Создание обработчика событий (а). Изменение элементов в ListBox (б) . . . . .	268
6.17	Главная форма приложения . . . . .	269
6.18	Скриншот главной формы приложения (ввод данных для решения СЛАУ) . . . . .	280
6.19	Скриншот главной формы приложения (ввод данных для обращения матрицы) . . . . .	280
6.20	Скриншот главной формы приложения (ввод данных для вычисления определителя) . . . . .	281
6.21	Внешний вид формы HandInputSLAU . . . . .	282
6.22	Внешний вид формы HandInputINV . . . . .	285
6.23	Подключение библиотек . . . . .	300
6.24	Выбор библиотек . . . . .	301
6.25	Ввод матриц для решения СЛАУ . . . . .	307
6.26	Решение СЛАУ и отчёт в таблице для задачи из рис. 6.25 . . . . .	307
6.27	Вывод таблиц результатов эксперимента: (а) в режиме «решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «обратная матрица» при использовании плохо обусловленных матриц девятого типа . . . . .	309
6.28	Вывод графика зависимости числа операций от порядка матрицы: (а) в режиме «Решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «Обратная матрица» при использовании плохо обусловленных матриц девятого типа . . . . .	310
6.29	Вывод графика зависимости времени выполнения от порядка матрицы: (а) в режиме «Решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «Обратная матрица» при использовании плохо обусловленных матриц девятого типа . . . . .	311
6.30	Вывод графика зависимости погрешности от порядка матрицы: (а) в режиме «Решение СЛАУ» при использовании матриц Гильберта и (б) в режиме «Обратная матрица» при использовании плохо обусловленных матриц девятого типа . . . . .	312
6.31	Эксперимент с вычислением определителя заданной матрицы: (а) Ввод матрицы и (б) Результат вычисления определителя . . . . .	313
7.1	Строчно ориентированная схема $\bar{L}U$ -разложения . . . . .	318
7.2	Столбцово ориентированная схема $\bar{L}U$ -разложения . . . . .	318
7.3	Сложение $n$ чисел методом сдваивания для $n = 8$ [9] . . . . .	321
7.4	$ij$ (слева) и $ji$ (справа) формы матрично-векторного умножения [9] . . . . .	323
7.5	Операция сложения в компьютере типа «регистр–регистр» [9] . . . . .	324
7.6	Столбцово ориентированная схема $\bar{L}U$ -разложения с отложенными модификациями ( $jki$ -алгоритм, см. с. 328) [9] . . . . .	325

7.7 Способ доступа к данным для $kij$ -формы (слева) и для $kji$ -формы (справа) $\bar{L}U$ -разложения. Обозначения: $\bar{L}$ , $U$ – вычисление закончено, обращений больше нет; $\boxtimes$ обозначает главный элемент ( $\Gamma$ Э); $\oslash$ – деление на $\Gamma$ Э (нормировка) [9] . . . . .	329
7.8 Способ доступа к данным для $jki$ -формы и для $jik$ -формы (слева) и для $ikj$ -формы и для $ijk$ -формы (справа) $\bar{L}U$ -разложения. Обозначения: $\bar{L}$ , $U$ – вычисление закончено, обращения больше не производятся; $\boxtimes$ обозначает главный элемент ( $\Gamma$ Э); $\oslash$ обозначает деление на $\Gamma$ Э (нормировка); $\emptyset$ – обращений не было [9] . . . . .	329
7.9 Алгоритмы скалярных произведений (слева) и столбцовый для обратной подстановки (справа) . . . . .	333
 8.1 Алгоритмы окаймления известной части $\bar{L}U$ -разложения: столбцовый (слева) и алгоритм скалярных произведений (справа) . . . . .	340
8.2 Доступ к данным в алгоритмах окаймления известной части разложения. $\bar{L}$ , $U$ здесь вычисление закончено, но обращения происходят. $A$ – обращений не было. Вычисляются: $j$ -й столбец матрицы $U$ и $j$ -я строка матрицы $\bar{L}$ . . . . .	341
8.3 Доступ к данным в алгоритмах окаймления неизвестной части разложения. $\bar{L}_{11}$ , $U_{11}$ – вычисление закончено, обращений больше нет. $L_{31}$ , $U_{13}$ здесь вычисление закончено, но обращения происходят. $A_{33}$ – обращений не было. Вычисляются: $j$ -й столбец $l_{3j}$ матрицы $\bar{L}$ и $j$ -я строка $u_{j3}^T$ матрицы $U$ . . . . .	342
8.4 Алгоритмы Донгарры–Айзенштата окаймления неизвестной части $\bar{L}U$ -разложения: алгоритм линейных комбинаций (слева) и алгоритм скалярных произведений (справа) . . . . .	343
 11.1 Линейная задача наименьших квадратов . . . . .	389
12.1 Включение априорных данных в линейную задачу НК . . . . .	398
12.2 Рекурсия МНК в стандартной ковариационной форме (схема Калмана). Этап 1 – обновление модели по наблюдениям. Этап 2 – экстраполяция модели между наблюдениями. Априорная модель даёт предсказание $\hat{z}$ для отклика $z$ объекта. Разность $\nu$ имеет смысл обновляющего процесса. Весовая матрица $K$ , умноженная на $\nu$ , обеспечивает обновление априорной модели по наблюдению $z$ , т. е. переход к апостериорной модели . . . . .	403

# Список таблиц

1	Влияние неуважительных пропусков на оценку . . . . .	20
2.1	Арифметические операции – иллюстрация для текста на с. 53 . . . . .	84
3.1	Свойства специальных элементарных матриц . . . . .	99
3.2	Поэтапное перемножение $L_4^{-1}(L_3^{-1}(L_2^{-1}L_1^{-1}))$ . . . . .	105
3.3	Поэтапное перемножение $\overline{U}_2^{-1}(\overline{U}_3^{-1}\overline{U}_4^{-1})$ . . . . .	107
4.1	Варианты задания на лабораторный проект № 2 . . . . .	151
5.1	Эффективное обращение верхней треугольной матрицы $U := R^{-1}$ . . . . .	179
5.2	Варианты задания на лабораторный проект № 3 . . . . .	199
7.1	Вычисления по алгоритму $jki$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом . . . . .	330
7.2	Вычисления по алгоритму $jik$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом . . . . .	330
7.3	Вычисления по алгоритму $ikj$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом . . . . .	331
7.4	Вычисления по алгоритму $ijk$ -формы для примера 7.3. Позиции ГЭ (без их реального выбора) показаны выделенным шрифтом . . . . .	332
7.5	Варианты задания на лабораторный проект № 4 . . . . .	336
8.1	Вычисления по алгоритмам на рис. 8.1 для примера 7.3. Позиции элемента-делителя столбца $\overline{L}$ показаны выделенным шрифтом . . . . .	341
8.2	Вычисления по алгоритмам на рис. 8.4 для примера 7.3. Позиции элемента-делителя столбца $\overline{L}$ показаны выделенным шрифтом . . . . .	344
8.3	Варианты задания на лабораторный проект № 5 . . . . .	347
9.1	Варианты задания на лабораторный проект № 7 . . . . .	369
10.1	Варианты задания на лабораторный проект № 6 . . . . .	386
11.1	Варианты задания на лабораторный проект № 8 . . . . .	394

# Библиографический список

## Основная литература

1. Бахвалов, Н. С. Численные методы / Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков. – М.: Наука, 1987.
2. Вержбицкий, В. М. Основы численных методов: учебное пособие для вузов / В. М. Вержбицкий. – М.: Вышш. шк., 2002, – 2-е изд., перераб. – М.: Вышш. шк., 2005.
3. Воеводин, В. В. Вычислительные основы линейной алгебры / В. В. Воеводин. – М.: Наука, 1977.
4. Воеводин, В. В. Численные методы алгебры. Теория и алгоритмы / В. В. Воеводин. – М.: Наука, 1966.
5. Самарский, А. А. Численные методы / А. А. Самарский, А. В. Гулин. – М.: Наука, 1989.
6. Семушин, И. В. Вычислительные методы алгебры и оценивания / И. В. Семушин. – Ульяновск: УлГТУ, 2011.

## Дополнительная литература

7. Годунов, С. К. Решение систем линейных уравнений / С. К. Годунов. – Новосибирск: Наука, 1980.
8. Икрамов, Х. Д. Численное решение линейных задач метода наименьших квадратов / Х. Д. Икрамов // Математический анализ. Итоги науки и техники, Т. 23. – М.: ВИНИТИ, 1985.
9. Орtega, Дж. Введение в параллельные и векторные методы решения линейных систем / Дж. Орtega. – М.: Мир, 1991.
10. Орtega, Дж. Введение в численные методы решения дифференциальных уравнений / Дж. Орtega, У. Пул. – М.: Наука, 1986.
11. Писсанецки, С. Технология разреженных матриц / С. Писсанецки. – М.: Мир, 1988.
12. Райс, Дж. Матричные вычисления и математическое обеспечение / Дж. Райс. – М.: Мир, 1984.
13. Стрэнг, Г. Линейная алгебра и ее применения / Г. Стрэнг. – М.: Мир, 1980.
14. Уидроу, Б. Адаптивная обработка сигналов / Б. Уидроу, С. Стирнз. – М.: Радио и Связь, 1989.
15. Фаддеев, Л. К. Вычислительные методы линейной алгебры / Л. К. Фаддеев, В. Н. Фаддеева. – М.: Физматгиз, 1963.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

---

16. Bierman, G. J. Factorization methods for discrete sequential estimation / G. J. Bierman. – New York, San Francisco, London: Academic Press, 1977.
17. Dorf, R. C. Modern control systems, Fifth Edition / Richard C. Dorf. – Reading, Massachusetts • Menlo Park, California • New York • Don Mills, Ontario • Wokingham, England • Amsterdam • Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan: Addison-Wesley Publishing Company, 1990.
18. Duff, I. S. Direct Methods for Sparse Matrices / I. S. Duff, A. M. Erisman, J. K. Reid. — Oxford: Clarendon Press, 1986.
19. Farebrother, R. W. Linear Least Squares / R. W. Farebrother. — New York and Basel: Marcel Dekker, Inc., 1988.
20. Fröberg, C. E. Introduction to Numerical Analysis / C. E. Fröberg. — Reading, Massachusetts · Menlo Park, California · London · Don Mills, Ontario: Addison-Wesley, 1969.
21. Hackbusch, W. Multi-grid methods and applications / W. Hackbusch. — New York: Springer Verlag, 1985.
22. McCormick, S. F. (Ed.) Multigrid methods / S. F. McCormick. – New York: Marcel Dekker, 1988.
23. Saad, Y. Iterative Solution of Linear Systems in the 20th Century / Y. Saad, H. Van der Vorst // *J. Comput. Appl. Math.*. – 2000. – No. 123. – P. 1–33.
24. Saad, Y. Iterative Methods for Sparse Linear Systems / Y. Saad. – 2nd ed. – SIAM, 2003.
25. Shewchuk, J. An Introduction to the Conjugate Gradient Method without the Agonizing Pain / J. Shewchuk. – Technical Report No. CMU-CS-94-125. – Carnegie Mellon University, 1994.
26. Van der Vorst, H. Iterative Krylov Methods for Large Linear Systems / H. Van der Vorst. – Cambridge: Cambridge University Press. – 2003.
27. Verkhovsky, B. Algorithm with Nonlinear Acceleration for a System of Linear Equations / B. Verkhovsky. – Research Report. – No. 76-WR-1. – Princeton University, 1976.
28. Veroy (now Verkhovsky), B. Convergence and Complexity of Aggregation–Disaggregation Algorithm / B. Veroy (now Verkhovsky). – Research Report. – No. CS-87-04. – New Jersey Institute of Technology, 1987.
29. Verkhovsky, B. Feedback Algorithm for the Single-facility Minisum Problem / B. Verkhovsky, Yu. Polyakov // *Annals of the European Academy of Sciences*. – 2003. – P. 127–136.
30. Watkins, D. S. The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods / D. S. Watkins. – SIAM, 2007.
31. Wesseling, P. An Introduction to Multigrid Methods / P. Wesseling. – Chichester: John Wiley & Sons, 1992.

## Литература по вопросам методики

32. Агентство Стратегических Инициатив РФ (АСИ РФ <http://www.asi.ru>). А. Современные технологии проектно-ориентированного образования [http://cdiorussia.ru/app/data/uploads/2013/-12/Gusev\\_ASI.pdf](http://cdiorussia.ru/app/data/uploads/2013/-12/Gusev_ASI.pdf),  
Б. Инициатива “Инженерного проектного образования CDIO” [http://cdiorussia.ru/app/data/uploads/2013/12/Presentation\\_CDIO\\_Gusev\\_Glavnaya.pdf](http://cdiorussia.ru/app/data/uploads/2013/12/Presentation_CDIO_Gusev_Glavnaya.pdf).
33. Апостолова, Н. А. О программометрическом подходе к оценкам программного обеспечения / Н. А. Апостолова, Б. С. Гольдштейн, Р. А. Зайдман // Программирование. – 1995. – С. 38–44.
34. Джонс, Дж. К. Методы проектирования. Пер. с англ. 2-е изд., доп. / Дж. К. Джонс. – М.: Мир, 1986.
35. Ренъи, А. Трилогия о математике / А. Ренъи. – М.: Мир, 1980.
36. Семушин, И. В. Модификация поведения студента и преподавателя инженерных дисциплин / И. В. Семушин, В. В. Угаров // Московское научное обозрение. – 2013. – № 9(37). – С. 3–8.
37. Семушин, И. В. Опыт проектно-ориентированного обучения в университетах Ульяновска / И. В. Семушин, В. В. Угаров, Ю. В. Цыганова, А. И. Афанасова, И. Н. Куличенко // Перспективные информационные технологии (ПИТ 2014): труды Международной научно-технической конференции / под ред. С. А. Прохорова. – Самара: Издательство Самарского научного центра РАН, 2014. – С. 436–438.
38. Семушин, И. В. Практикум по методам оптимизации – Компьютерный курс / И. В. Семушин. – Ульяновск: УлГТУ, 2003.
39. Семушин, И. В. Практикум по методам оптимизации. Компьютерный курс: учебное пособие для вузов / И. В. Семушин. – 3-е изд, перераб. и доп. – Ульяновск: УлГТУ, 2005.
40. Семушин, И. В. Методы вычислений с использованием МАТЛАБ: учебно-методическое пособие / И. В. Семушин, Ю. В. Цыганова, А. И. Афанасова. – Ульяновск: УлГУ, 2014. – 88 с.
41. Угаров, В. В. Компьютерные модели и пакеты программ в проектно-ориентированном обучении / В. В. Угаров. Диссертация канд. техн. наук. – Ульяновск: УлГУ, 2005.
42. Холстед, М. Х. Начала науки о программах / М. Х. Холстед. Пер. с англ. В. М. Юфы. – М.: Финансы и статистика, 1981.
43. The CDIO Initiative <http://www.cdio.org/>. CDIO [Conceive - Design - Implement - Operate] – Современный подход к инженерному образованию <http://cdiorussia.ru/>.
44. Dewey, J. The school and society / J. Dewey. – Chicago: University of Chicago, 1900.
45. Dumestre, J. Using computer gaming technologies to make training and education software more effective / Jeanie Dumestre // Proceedings of SimTecT-97. – 1997. – Р. 447–451.
46. Fitzpatrick, J. M. Computer Programming with Matlab / J. Michael Fitzpatrick & Ákos Lédeczi. – Fitzle LLC, 2013.
47. Jonassen, D. H. Computers in the schools: Mindtools for critical thinking / D. H. Jonassen. – College Park, PA: Penn State Bookstore, 1994.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

---

48. Kilpatrick, W. H. The project method / W. H. Kilpatrick // *Teachers College Record*. – 1918. – No. 19. – P. 319–335.
49. Liu, M. Middle school students as multimedia designers: a project-based learning approach / Min Liu, Yu-Ping Hsiao // *Jl. of Interactive Learning Research*. – 2002. – No. 13(4). – P. 311–337.
50. McDermott, K. J., et al. Project-based Teaching in Engineering Programs / Kevin J. McDermott, Andrew Nafalski and Özdemir Göl // *37th ASEE/IEEE Frontiers in Education Conference*, Session S1D-11, October 10–13, 2007, Milwaukee, WI.
51. Measuring the Mathematics Problem / Report by • The Learning and Teaching Support Network (Maths, Stats & OR) • The Institute of Mathematics and its Applications • The London Mathematical Society • The Engineering Council, Published by the Engineering Council: June 2000.
52. Semoushin, I. V. The frontal competitive approach to teaching computational mathematics / Innokenti V. Semoushin. *Presented for The 9rmth International Congress on Mathematics Education (ICME-9)*. July 31 – August 6, 2000, Tokyo/Makuhary, Japan.
53. Semoushin, I. V. FCA + PBL = behavior modification in learning operations research / Innokenti V. Semoushin. *MAA Session on Teaching Operations Research in the Undergraduate Classroom*. January 7–10, 2004, Phoenix, Arizona.
54. Semoushin, Innokenti V. Computational and soft skills development through the project based learning / Innokenti V. Semoushin, Julia V. Tsyganova, and Vladimir V. Ugarov // *Lecture Notes in Computer Science*. – 2003. – Vol. 2658, Pt. 2. – P. 1098–1106.
55. Semoushin, I. V. The frontal competitive approach to teaching computational mathematics / Innokenti V. Semoushin // *The 2nd International Conference on the Teaching Mathematics (at the undergraduate level)*, 1–6 July 2002, Crete, Greece. – 2002. – , CD-ROM Proceedings. – File ID265.
56. Semoushin, I. V. Project based learning in computational science and engineering / I. V. Semoushin, Ju. V. Tsyganova, V.V. Ugarov // *Математические методы и информационные технологии в экономике, социологии и образовании*. Сборник научных трудов: В. И. Левин (ред.) Международная конференция «Математические методы и информационные технологии в экономике, социологии и образовании». – 2003. Пенза: Пензенский Дом Знаний. – С. 244–245.
57. Tackling the Mathematics Problem / London Mathematical Society, Institute of Mathematics and its Applications, Royal Statistical Society, October 1995.
58. Thomas, J. W. (2000). A review of research on project-based learning / J. W. Thomas // – Retrieved December, 10, 2000, from: <http://www3.autodesk.com/adsk/index/0,,327082-123112,00.-html>.

# Предметный указатель

## А

- академический программный продукт, АПП  
(academic software product) 41
- алгоритм
  - векторных сумм (vector sums algorithm) 331
  - Джозефа скаляризованный (scalarized Joseph algorithm) 406
  - Донггарры–Айзенштата (Dongarra–Eisenstat algorithm) 342
  - Калмана скаляризованный (scalarized Kalman algorithm) 405
  - Калмана стабилизированный (stabilized Kalman algorithm) 406
  - Калмана стандартный (standard Kalman algorithm) 403
  - $L\bar{U}$  компактной схемы Краута (Crout compact schemata,  $L\bar{U}$  algorithm) 95, 97
  - $\bar{L}U$  компактной схемы Краута (Crout compact schemata,  $\bar{L}U$  algorithm) 97
  - Крылова с предобуславливанием (Krylov algorithm with preconditioning) 366
  - метода Жордана (Jordan method algorithm) 97, 99
  - обратной подстановки (back substitution algorithm) 331
  - Поттера (Potter algorithm) 407
  - прямой подстановки (forward substitution algorithm) 333
  - столбцовый (column algorithm) 331
  - фильтра Поттера (Potter filter algorithm) 407
  - Хаусхолдера столбцово ориентированный (column based Hausholder algorithm) 175
  - Хаусхолдера строчно ориентированный (row based Hausholder algorithm) 175

## Б

- базовая арифметика C# (C# basic arithmetics) 53
- бidiагонализация квадратной матрицы (square matrix bidiagonalization) 188

## В

- вариационные методы (variation methods) 360
- ведущий столбец
  - в преобразовании Гивенса (leading column in Givens transform) 181
  - в преобразовании Хаусхолдера (leading column in Hausholder transform) 173
- векторные регистры (vector registers) 323
- выбор ведущего (главного) элемента (pivoting) 93
  - по активной подматрице (active submatrix based pivoting) 92
  - по столбцу (column based pivoting) 91, 318
  - по строке (row based pivoting) 91
  - разреженной матрицы оптимальный (optimal sparse matrix pivoting) 382

## Г

- Галилей (Galilei) 5, 6
- Гаусс (Gauss) 168
- Грама–Шмидта (Gram–Schmidt)
  - ортогонализация (Gram–Schmidt orthogonalization) 190, 191
  - – модифицированная (modified Gram–Schmidt orthogonalization) 192

## Д

- Джебран (Gibran) 5, 6
- диагональное преобладание (diagonal prevaling) 136

## З

- задача
  - линейных наименьших квадратов (linear least squares problem) 168
  - $QR$ -разложения матрицы (problem of matrix  $QR$ -decomposition) 192
- заполнение локальное (local filling) 381
- защита проекта (project defense) 50

# ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

---

## И

- инициатива CDIO (CDIO initiative) 30
- исключение
  - по столбцам (column based elimination) 92
  - по строкам (row based elimination) 93
  - гауссово по строкам (Gauss row based elimination with row based pivoting) 93
  - полное (full elimination) 97
- итерационный метод (ИМ) (iterative method) 351, 360
  - Зейделя (iterative Seidel method) 353
  - многошаговый (multi step iterative methods) 354
  - нестационарный (non-stationary IM) 352, 355
  - неявный (indirect IM) 351
  - одношаговый (one-step iterative methods) 354
  - стационарный в терминах погрешности (one-step stationary IM in terms of errors) 356
  - стационарный (stationary IM) 352, 355
  - явный (direct method) 351
  - Якоби (iterative Jacobi method) 353
- информационная форма последовательного МНК (information form of sequential LS method) 401
- итерационная формула (iteration formula) 352

## К

- квадратичная (квадратическая) форма (quadratic form) 135
- квадратный корень матрицы (matrix square root) 136
- каноническая форма одношагового ИМ (canonical form of one-step iterative method) 355
- ковариация (covariation) 397
- компактная схема (compact schemata) 94
  - Краута (Crout compact schemata) 94
- конвейеризация (conveyerization) 320
- коэффициенты корреляции (correlation coefficients) 136
- критерий
  - качества квадратический performance criterion) 168
  - остановки (stopping criterion) 352
  - Сильвестера (Silvester criterion) 136

## Л

- Лежандр (Legendre) 168

## М

- матрица
  - идемпотентная (idempotent matrix) 171
  - информационная (information matrix) 397
  - ковариационная (covariation matrix) 397
  - лидирующая (leading matrix) 355
  - обратная (inverse matrix) 101
  - ортогональная (orthogonal matrix) 166
  - перестановок (perturbation matrix) 90
    - элементарная (elementary perturbation matrix) 90
  - переходная погрешности (error transition matrix) 357
  - плохо обусловленная (ill-conditioned matrix) 107, 359
  - положительно определенная (positive definite matrix) 135
  - псевдообратная (pseudo-inverse matrix) 168, 389
  - симметрическая (symmetric matrix) 171
  - Хаусхолдера (Householder matrix) 171
  - элементарная (elementary matrix) 97
    - специальная (special elementary matrices) 98
- метод
  - верхней релаксации (Successive Over-Relaxation, SOR method) 355
  - Гаусса, полный шаг (full step of Gauss method) 88
  - Гаусса, прямой ход (forward move of Gauss method) 89
  - Гаусса, обратный ход (back move of Gauss method) 90
  - делинеаризации (delinearization method) 366
  - минимальных невязок (minimum residual method) 360
    - поправок (minimum correction method) 361
  - многосеточный (multi-grid method) 366
  - наименьших квадратов (МНК) (least squares method) 168, 196
  - проектов (project method) 29
  - простой итерации (simple iteration method) 355
  - Ричардсона (Richardson method) 355
  - скорейшего спуска (quickest descent method) 363
  - сопряженных градиентов (conjugate gradients method) 363
  - Юнга (Young method) 355
- методика Холстеда (Halstead method) 43
- миноры, главные (main minors) 88

модификации

- немедленные (immediate modifications) 327
- отложенные (delayed modifications) 328

## H

направляющий вектор (direction vector) 170

невязка (residual) 168, 360

норма вектора (vector norm) 357

– обобщенная (generalized vector norm) 357

норма матрицы типа «бесконечность»  
( $\infty$  matrix norm) 111

нормализованная погрешность (normalized error)  
397

нормализованные экспериментальные данные  
(normalized experimental data) 396

нормальная система (normal system) 168, 196, 389

нормальное псевдорешение (normal pseudo-  
solution) 169, 196, 389

нормальные уравнения (normal equations) 168,  
196, 389

нормировка (norming) 87

## O

обновление

– системы (system update) 88

обратная подстановка (back substitution) 331

обращение верхней треугольной матрицы  
(inverting upper triangular matrix) 177

окаймление (bordering) 339

– известной части LU-разложения (bordering the  
known part of LU decomposition) 340

– неизвестной части LU-разложения (bordering  
the unknown part of LU decomposition)  
342

оптимальный итерационный параметр (optimal  
iteration parameter) 358

ортогонализация (orthogonalization) 190

отраженный вектор (reflected vector) 170

оценка качества программных продуктов  
(software project evaluation technique) 39

## П

переопределенная система (overdetermined  
system) 196

перестановка неявная (indirect interchanging) 318

погрешность

– методическая (method error) 351

– округления (roundoff error) 352

– трансформированная (transform error) 352

подстановка

– прямая (forward substitution) 89, 317

– обратная (back substitution) 89, 170, 317, 331

поправка ИМ (correction of IM) 361

порядок оценивания проектов (instruction for  
project evaluation) 49

предупреждение

– о термине  $L\bar{U}^{-1}$ -«разложение» ( $L\bar{U}^{-1}$   
«decomposition» term precaution) 100

– о термине  $\bar{L}^{-1}\bar{U}$ -«разложение» ( $\bar{L}^{-1}\bar{U}$   
«decomposition» term precaution) 101

преобразование

– Хаусхолдера (Hausholder transform) 170

– Гивенса (Givens transform) 179

– – строчно-ориентированное (row based Givens  
transform) 183

– – столбово-ориентированное (column based  
Givens transform) 183

проекто-ориентированная методика, преимущества  
(project based philosophy) 33

проекто-ориентированное обучение (project based  
learning) 29, 30, 39

проекция вектора (vector projection) 170

пространство

– Крылова (Krylov space) 365

процедура ортогонализации системы векторов  
(vector system orthogonalization procedure) 190

## P

разложение

– LU (LU-decomposition) 87

–  $\bar{L}\bar{U}$  ( $\bar{L}\bar{U}$ -decomposition) 89

– – гауссовым исключением по строкам (Gauss  
 $\bar{L}\bar{U}$  decomposition with row based  
elimination) 94

– – – жордановым исключением,  $\bar{L}\bar{U}^{-1}$   
«разложение» ( $\bar{L}\bar{U}^{-1}$  decomposition)  
 $A = \bar{L}\bar{U}$  by Jordan method) 101

– – с выбором главного элемента ( $\bar{L}\bar{U}$   
decomposition with pivoting) 92

–  $\bar{L}\bar{U}$  ( $\bar{L}\bar{U}$ -decomposition) 89, 90

– – с выбором главного элемента ( $\bar{L}\bar{U}$   
decomposition with pivoting) 92

– Холесского (Cholesky decomposition) 137

– – нижнее треугольное (lower triangular Cholesky  
decomposition) 137

# ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

---

- — — без квадратных корней (square root free lower triangular Cholesky decomposition) 138
- — верхнее треугольное (upper triangular Cholesky decomposition) 138
- — — без квадратных корней (square root free upper triangular Cholesky decomposition) 139
- расширенная система (extended system) 398
- расщепление системы (system splitting) 400
- решение
- задачи наименьших квадратов (МНК-решение) (least squares solution) 388
- одновременное (LS-simutaneous solution) 390
- последовательное (LS-sequential solution) 400

## C

- свойство
  - идемпотентности (idempotency feature) 91
  - несмещенности (unbiasedness property) 397
  - ортогональности (orthogonality feature) 91
- скалярное произведение (scalar product) 331
- скалярный параметр (scalar parameter) 351
- спектральный радиус матрицы (spectral radius of matrix) 357
- степень параллелизма (parallelization power) 320
  - средняя (mean parallelization power) 320
- специализированная программа для автоматического определения качественных характеристик программных продуктов (customized application for automatic academic code review) 45
- стратегии выбора ведущего (главного) элемента (pivoting strategies) 91
- сходимость итерационного метода (convergence of iterative method) 357

## T

- таблица множителей (table of multipliers) 99
- теорема
  - о верхнем треугольном разложении Холесского (upper triangular Cholesky decomposition) 138
  - о нижнем треугольном разложении Холесского (lower triangular Cholesky decomposition) 137
  - о триангуляции матрицы по методу Хаусхолдера (matrix triangularization by

- Hausholder transform) 172, 173
- о триангуляции матрицы по методу Гивенса (matrix triangularization by Givens transform) 181
- требование к реализации (implementation requirement) 93
- триада (triada) 318
- триангуляция матрицы
  - метод Хаусхолдера (Hausholder) 172, 173
  - метод Гивенса (Givens) 181
- тридиагонализация симметрической матрицы (symmetric matrix tridiagonalization) 187

## У

- упакованная форма матрицы (packed matrix forms) 379
- уравнение
  - ведущее (pivot equation) 87
- условие полного столбцового ранга (full column rank condition) 168

## Ф

- фильтр
  - Калмана (Kalman filter) 403
  - — обычновенный (the conventional Kalman filter) 403
  - — стандартный ковариационный (standard Kalman covariance filter, SCF) 403
  - Поттера квадратно-корневой (Potter Square Root Filter) 407
- фронтально-состязательный подход (frontal competitive approach) 34

## Х

- хранение матриц одномерное (linear (one dimension) matrix storing) 146

## Э

- элиминативная форма обратной матрицы (eliminative form of inverse matrix) 102
- экономия памяти (saving memory) 89
- элемент
  - ведущий (pivot) 87
  - главный (pivot) 90
  - допустимый (feasible element) 381

Учебное издание

**Семушин** Иннокентий Васильевич  
**Цыганова** Юлия Владимировна  
**Воронина** Валерия Вадимовна  
**Угаров** Владимир Васильевич  
**Афанасова** Анастасия Игоревна  
**Куличенко** Илья Николаевич

**ВЫЧИСЛИТЕЛЬНАЯ ЛИНЕЙНАЯ АЛГЕБРА  
В ПРОЕКТАХ НА C#**

Учебное пособие

Редактор М. В. Теленкова

ЛР № 020640 от 22.10.97

Оригинал-макет изготовлен И. В. Семушиным в системе L<sup>A</sup>T<sub>E</sub>X2<sup>C</sup>

Подписано в печать 16.12.2014. Формат 60×84/16. Усл. печ. л. 25,11.  
Гарнитура Computer Modern. Тираж 150 экз. Заказ № 6. ЭИ № 366.

Ульяновский государственный технический университет  
432027, г. Ульяновск, ул. Северный Венец, д. 32.  
ИПК «Венец» УлГТУ, 432027, г. Ульяновск, ул. Северный Венец, д. 32.

# Вычислительная линейная алгебра в проектах на C#

Книга охватывает базовые алгоритмы вычислительной линейной алгебры (ВЛА) и ориентирует на их изучение методом проектов.

Предлагаемые авторами проекты содержат более 250 индивидуальных заданий по основным темам ВЛА в трёх частях: «Стандартный курс», «Повышенный курс» и «Специальный курс».

Книга предназначена для студентов и аспирантов, обучающихся на факультетах информационных и вычислительных технологий.

ISBN 978-5-9795-1342-3



9 7 8 5 9 7 9 5 1 3 4 2 3

